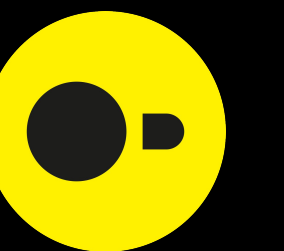


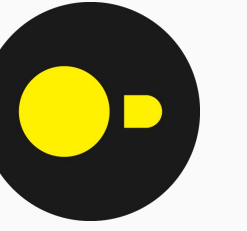


Storage and Encryption in DuckDB

Lotte Felius, January 2026

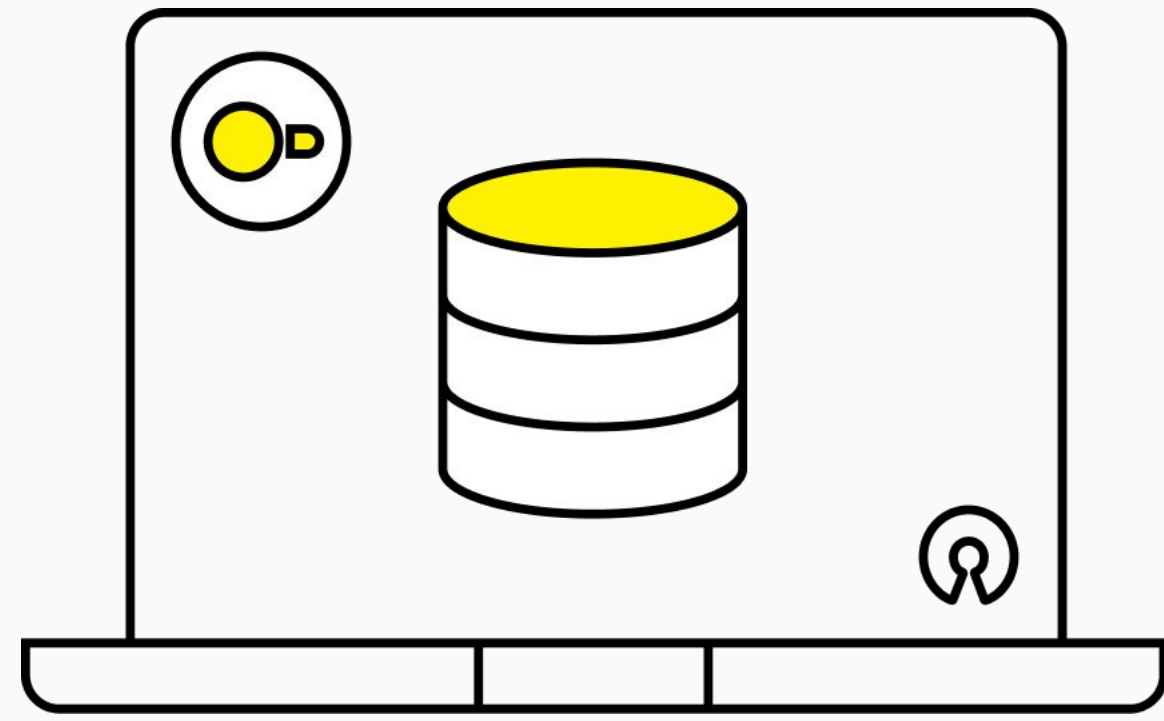


Who am I?



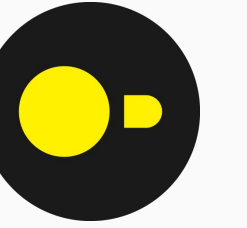
- MSc @ University of Amsterdam
- 2022: MSc thesis @ Citus team (Microsoft)
- 2023-2025: CWI
- Now: Software Engineer at DuckDB Labs





**How Can DuckDB
Secure Your Data?**

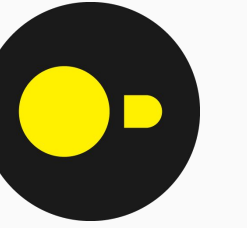
Data-at-rest encryption



Data-at-rest encryption: files on disk are encrypted

Parquet encryption (since v0.10.0)

Data-at-rest encryption



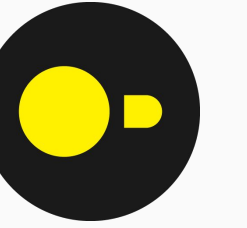
Data-at-rest encryption: files on disk are encrypted

Parquet encryption (since v0.10.0)

```
PRAGMA add_parquet_key('key128', '0123456789112345');
```

```
COPY tbl TO 'tbl.parquet'  
(ENCRYPTION_CONFIG { footer_key: 'key128' });
```

Data-at-rest encryption



Data-at-rest encryption: files on disk are encrypted

Database file encryption (since v1.4.0)

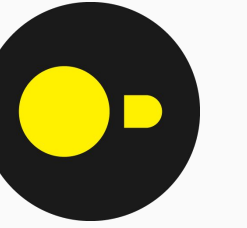
MEMORY

.DUCKDB

.DUCKDB.WAL

.TMP/ .BLOCK

Data-at-rest encryption



Data-at-rest encryption: files on disk are encrypted

Database file encryption (since v1.4.0)

MEMORY

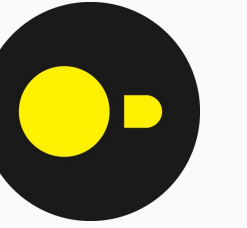
.DUCKDB

.DUCKDB.WAL

.TMP / .BLOCK



Data-at-rest encryption

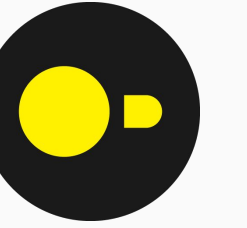


Data-at-rest encryption: files on disk are encrypted

Database file encryption (since v1.4.0)

```
ATTACH 'encrypted.duckdb' AS encrypted (  
    ENCRYPTION_KEY 'asdf',  
    ENCRYPTION_CIPHER 'GCM'  
);
```

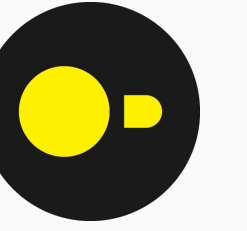
Advanced Encryption Standard



DuckDB uses the **Advanced Encryption Standard (AES)**

- **Galois Counter Mode (AES-GCM)**
- **Counter Mode (AES-CTR)**

Advanced Encryption Standard



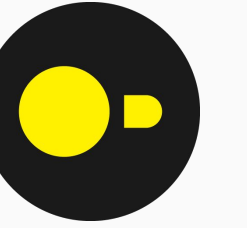
DuckDB uses the **Advanced Encryption Standard (AES)**

- **Galois Counter Mode (AES-GCM)**
- **Counter Mode (AES-CTR)**

This is *randomized* encryption

- Identical plaintexts yield different ciphertexts

Advanced Encryption Standard



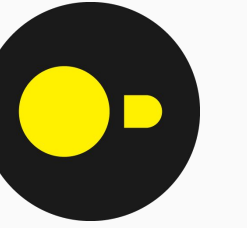
DuckDB uses the **Advanced Encryption Standard (AES)**

- **Galois Counter Mode (AES-GCM)**
- **Counter Mode (AES-CTR)**

This is *randomized* encryption

- Identical plaintexts yield different ciphertexts

AES-GCM



- Industry Standard
- Number Only Used Once (**nonce**) / Initialization Vector (**IV**)
- Computes a **tag**

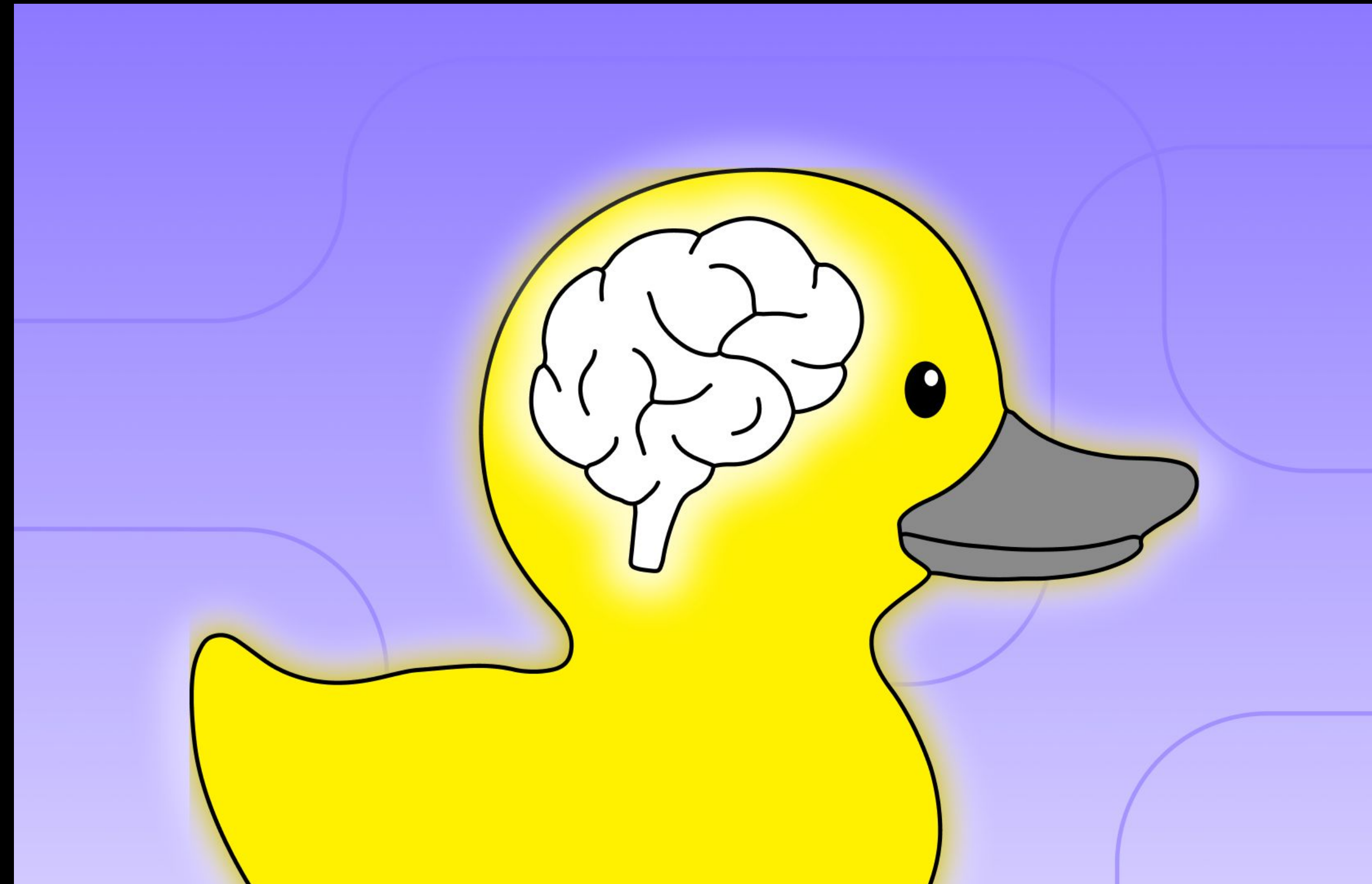
NONCE/IV
(12 BYTES)

ENCRYPTED TEXT

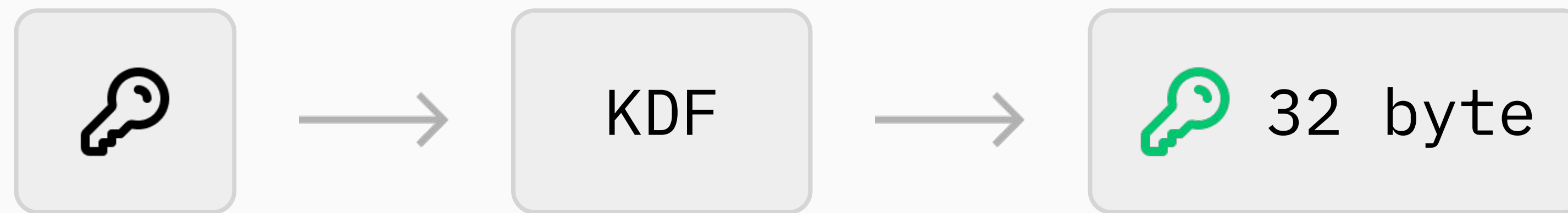
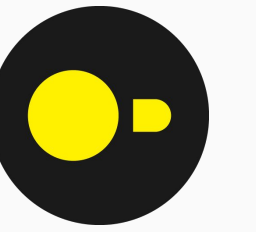


TAG
(16 BYTES)

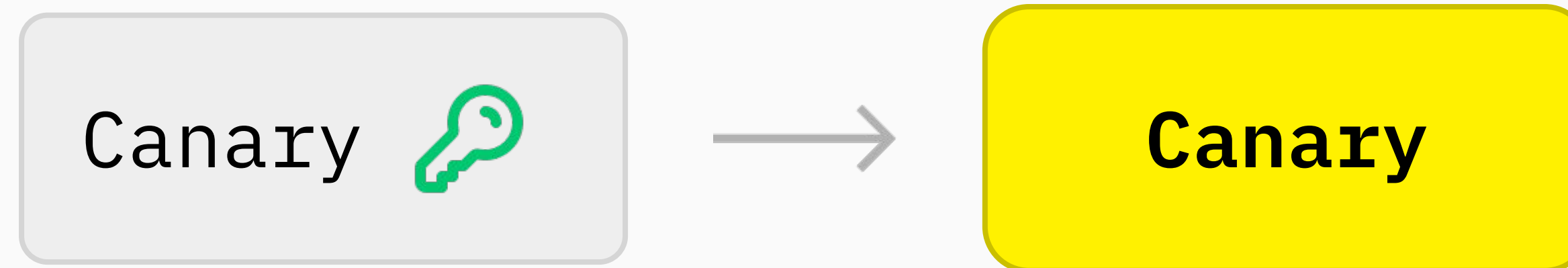
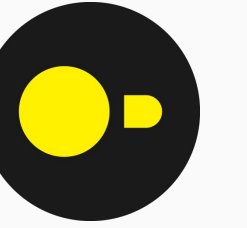
**How Do We
Handle the
Encryption
Keys?**



DuckDB uses Key Derivation



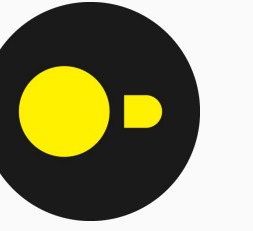
Is the correct key used?



If canary is **correct** → key is correct!

If **incorrect** → we error out early

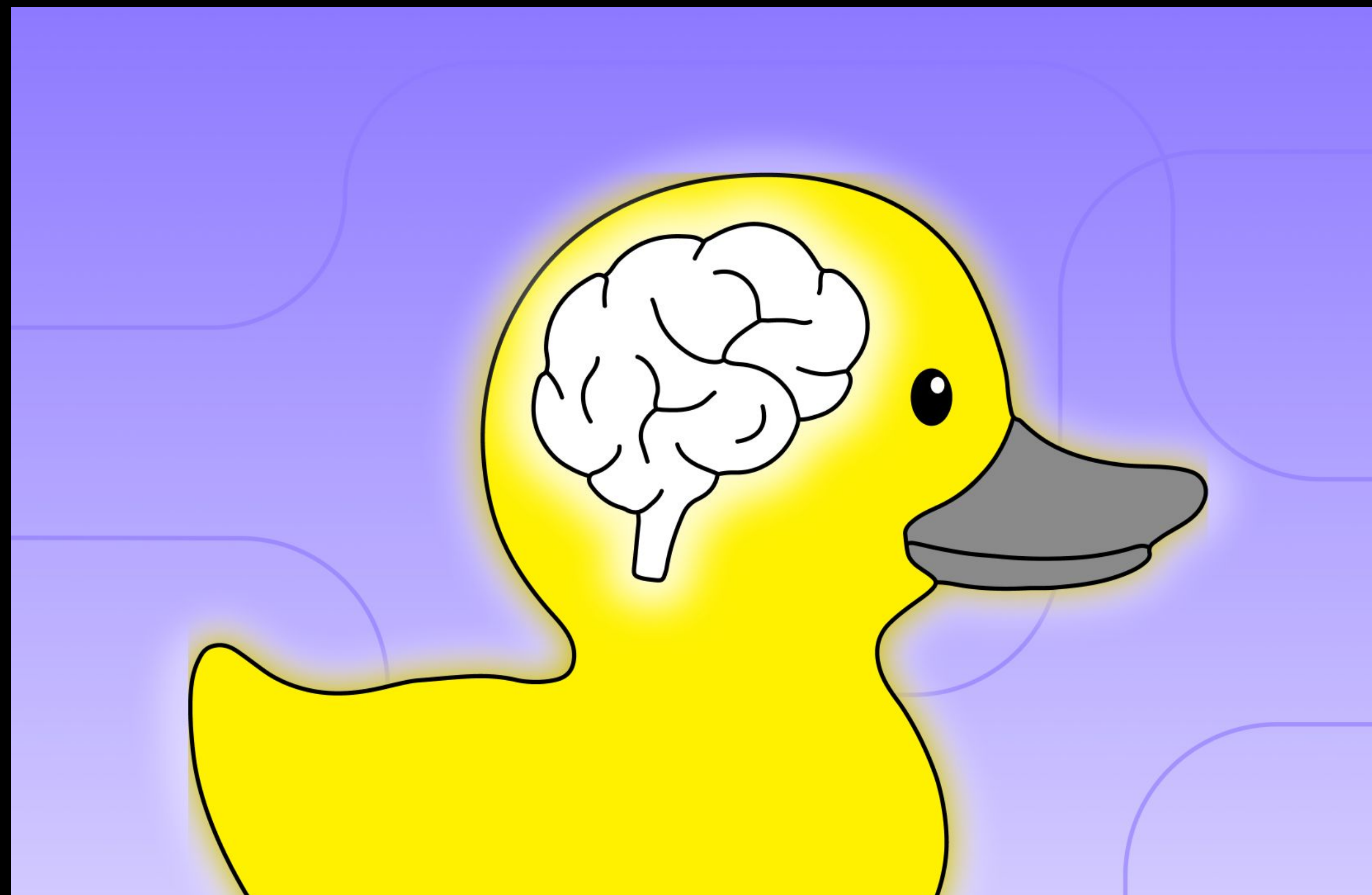
Encryption key management



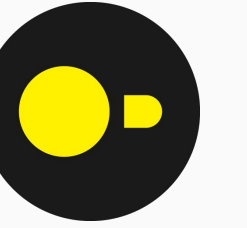
Secure Encryption Key Cache

- We **lock** the memory region
- The key is **wiped**

DuckDB Storage Overview



DuckDB (database) headers



file.db

MAIN HEADER

4KB

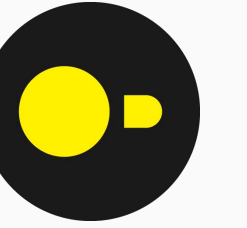
DATABASE HEADER (1)

4KB

DATABASE HEADER (2)

4KB

DuckDB (database) headers



file.db

MAIN HEADER

4KB

DATABASE HEADER (1)

4KB

DATABASE HEADER (2)

4KB

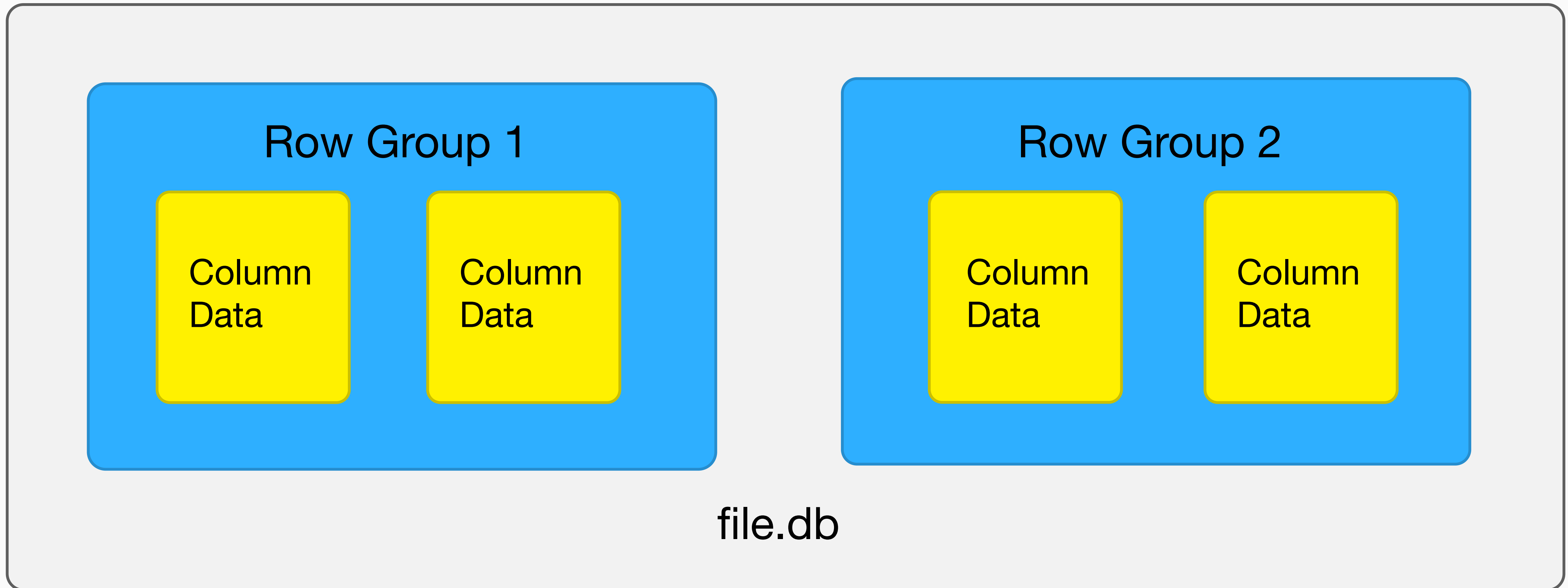
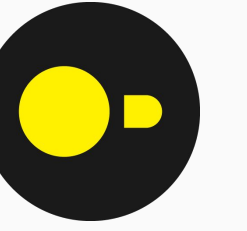
Main Header

(CHECKSUM)
DUCK
storage version: 64
(FLAGS)
v1.4.3
d1dc88f950
(ENCRYPTION METADATA)
(DATABASE UUID)
(ENCRYPTION CANARY)
(ENCRYPTION IV)
(ENCRYPTION TAG)

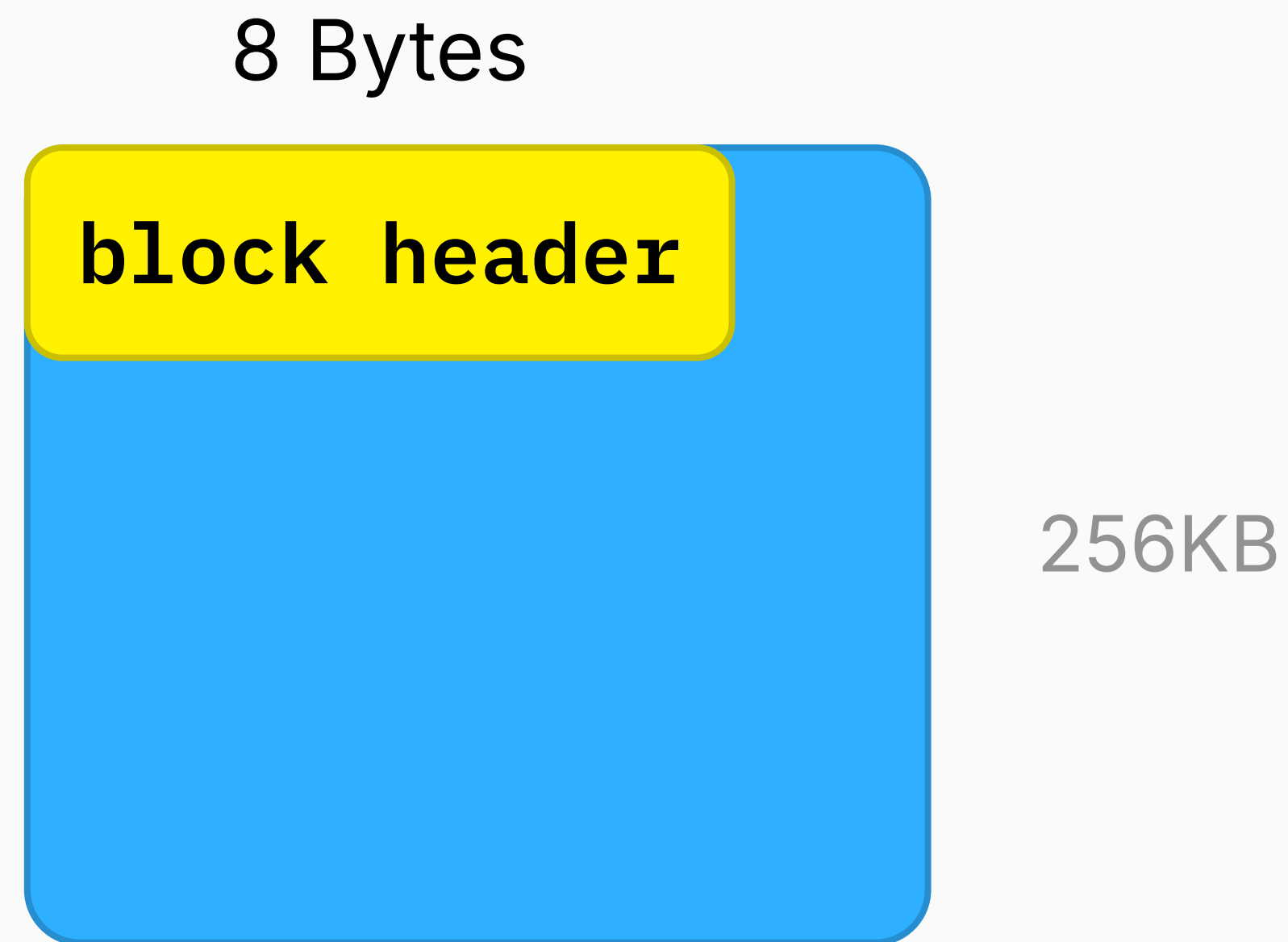
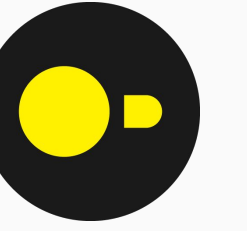
Database Header 1&2

(CHECKSUM)
iteration: 0
meta_block: -1
free_list: -1
block_count: 0
block_alloc_size: 262144
vector_size: 2048
serialization_compat: v1.0.0

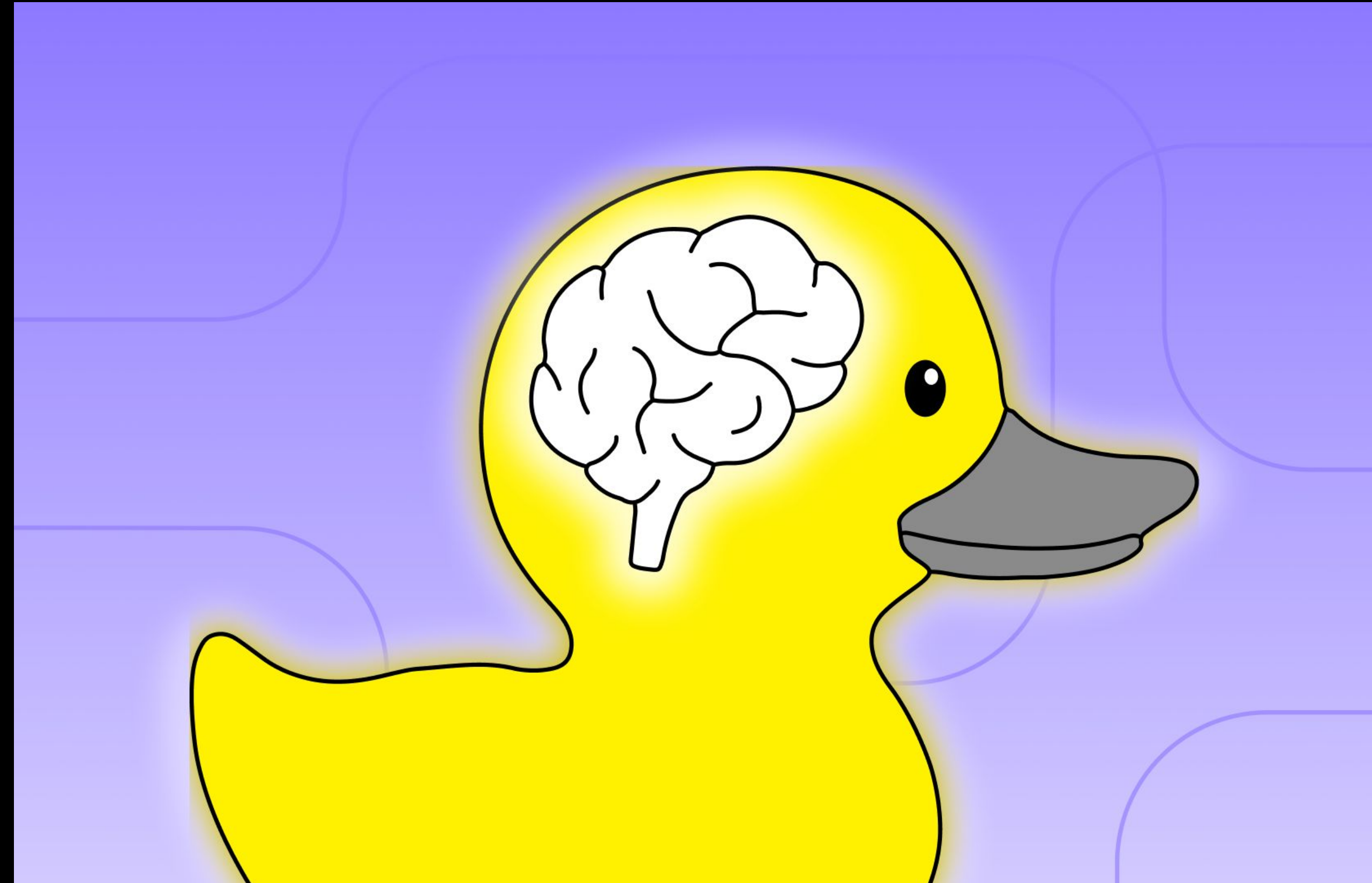
The storage structure of DuckDB



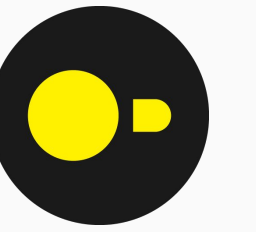
Each chunk is divided into 256kb blocks



Main Database Encryption



The main header remains plaintext



file.db

MAIN HEADER

4KB

DATABASE HEADER (1)

4KB

DATABASE HEADER (2)

4KB



encrypted.db

MAIN HEADER

4KB

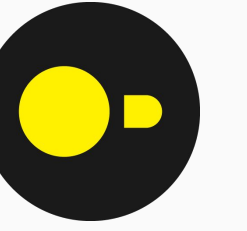
 **DATABASE HEADER (1)**

4KB

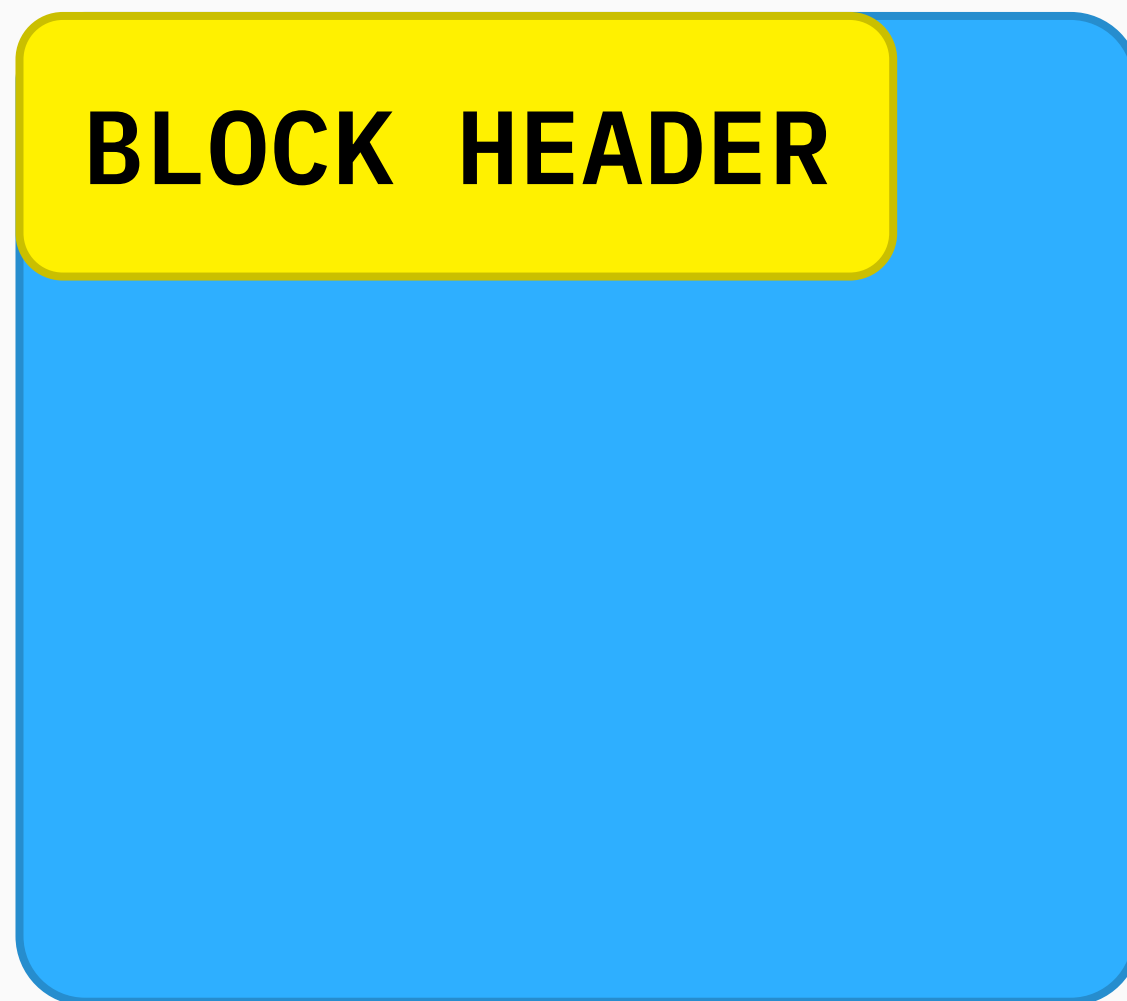
 **DATABASE HEADER (2)**

4KB

DuckDB encrypts in blocks to minimize overhead

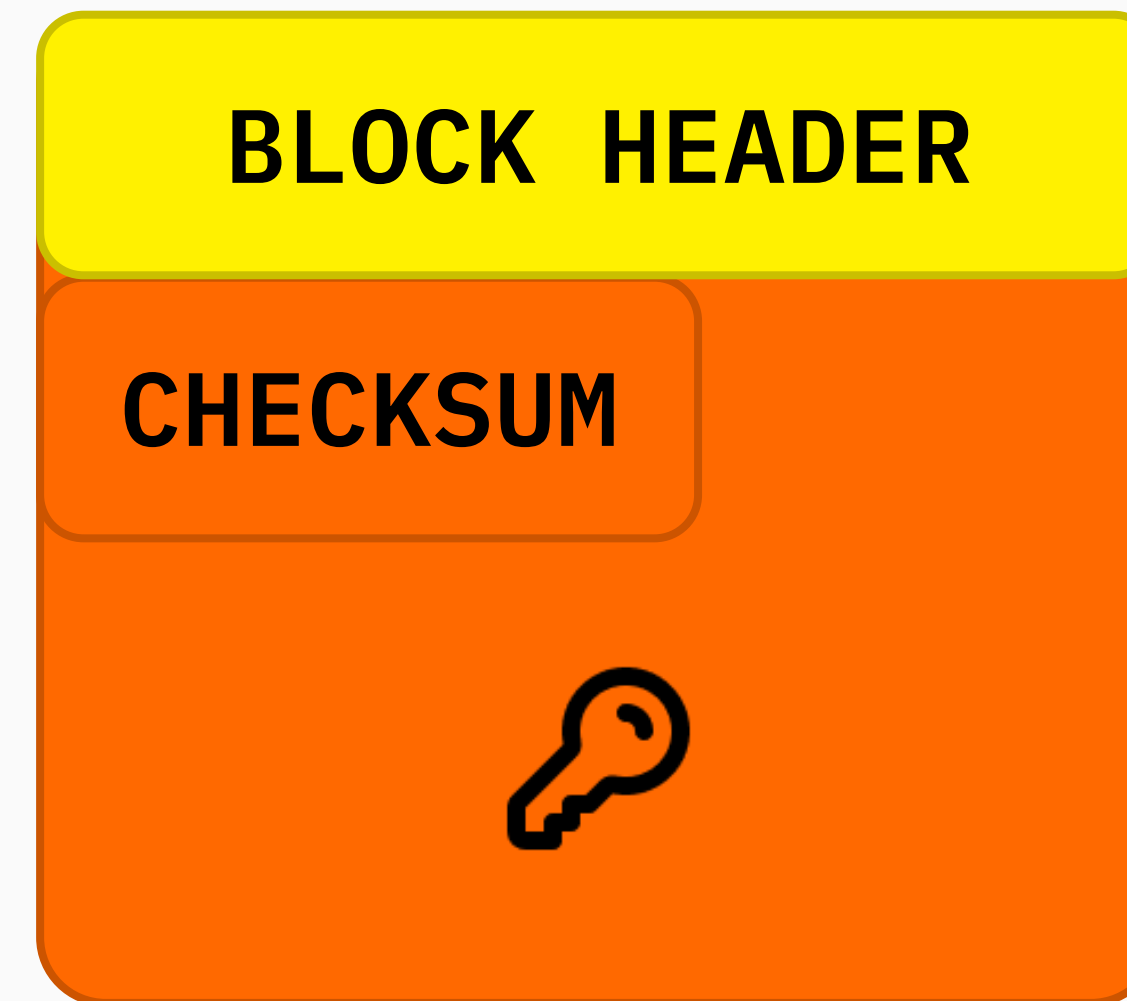


8 Bytes



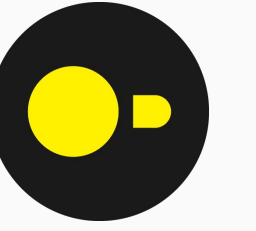
256KB →

8 Bytes + 32 Bytes

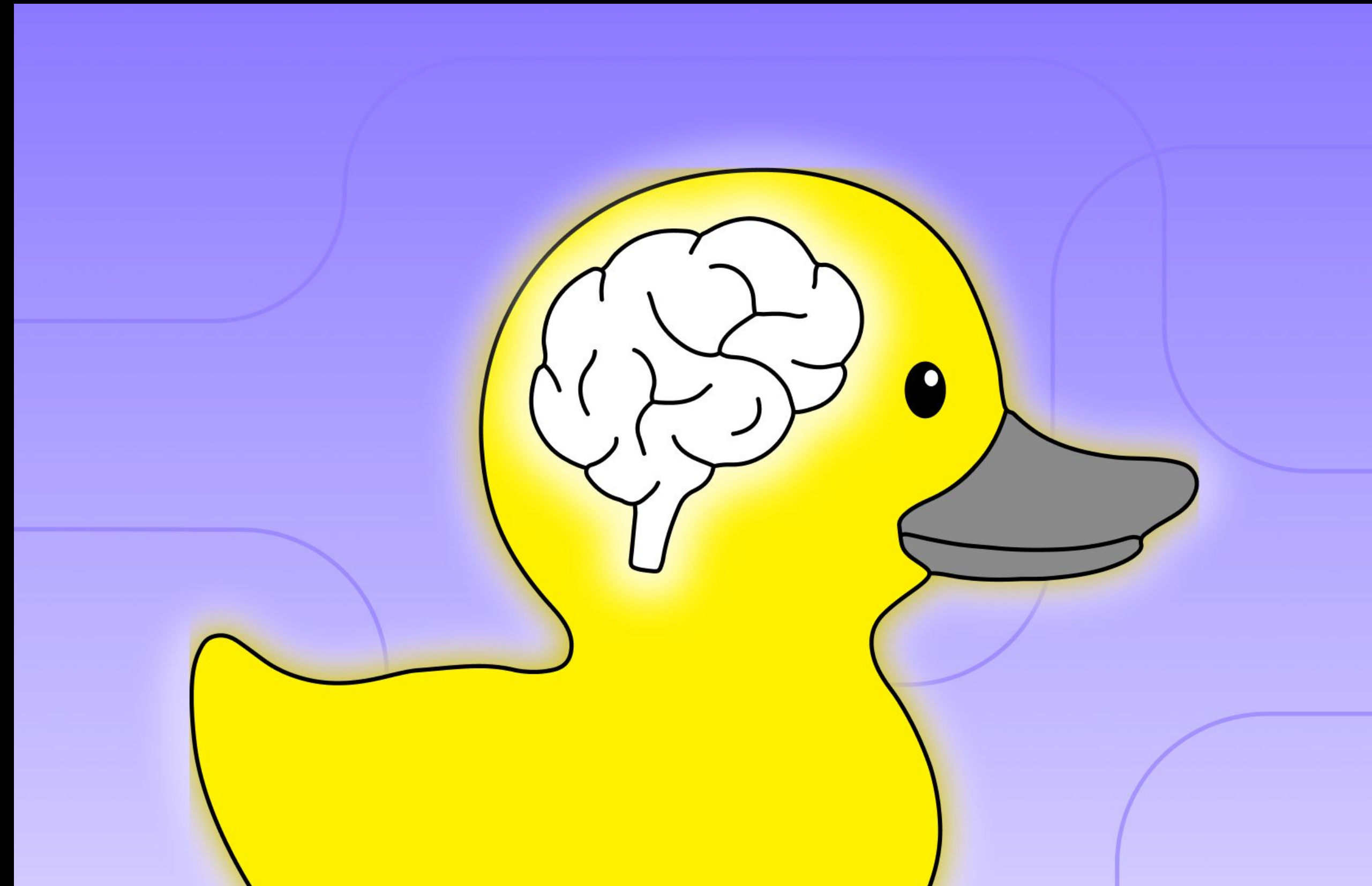


256KB

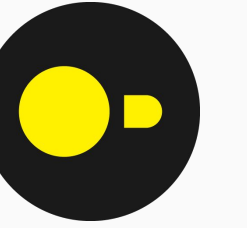
Block header for encrypted blocks



Write-Ahead Log (WAL)

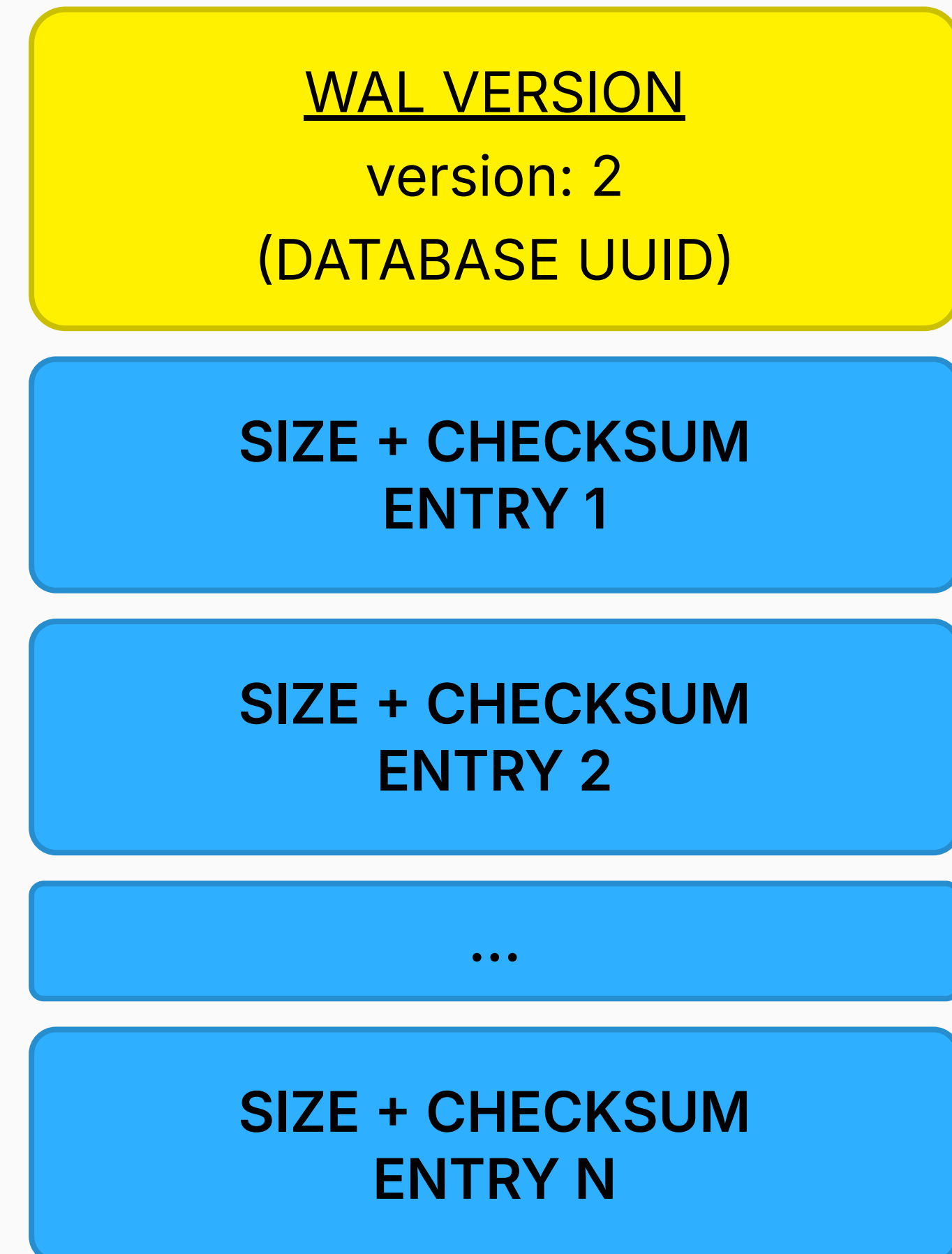


What is a Write Ahead Log (WAL)?

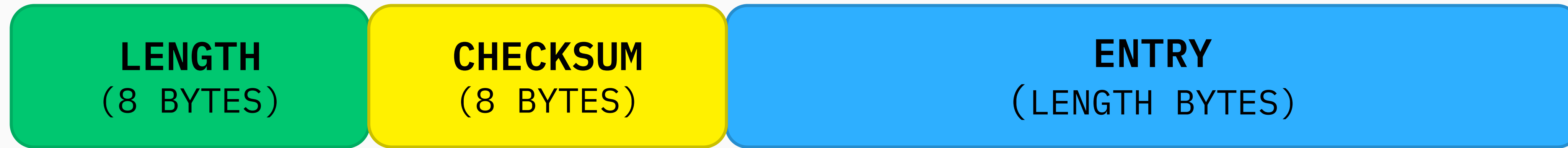
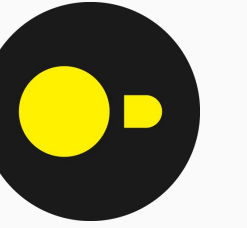


- **Write-Ahead-Log (WAL)** is a **crash recovery** mechanism
- Also important for **ACID** transactions
- Can be used to increase **performance**

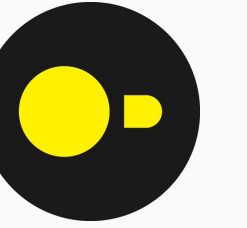
file.db.wal



Each WAL entry contains a length and checksum

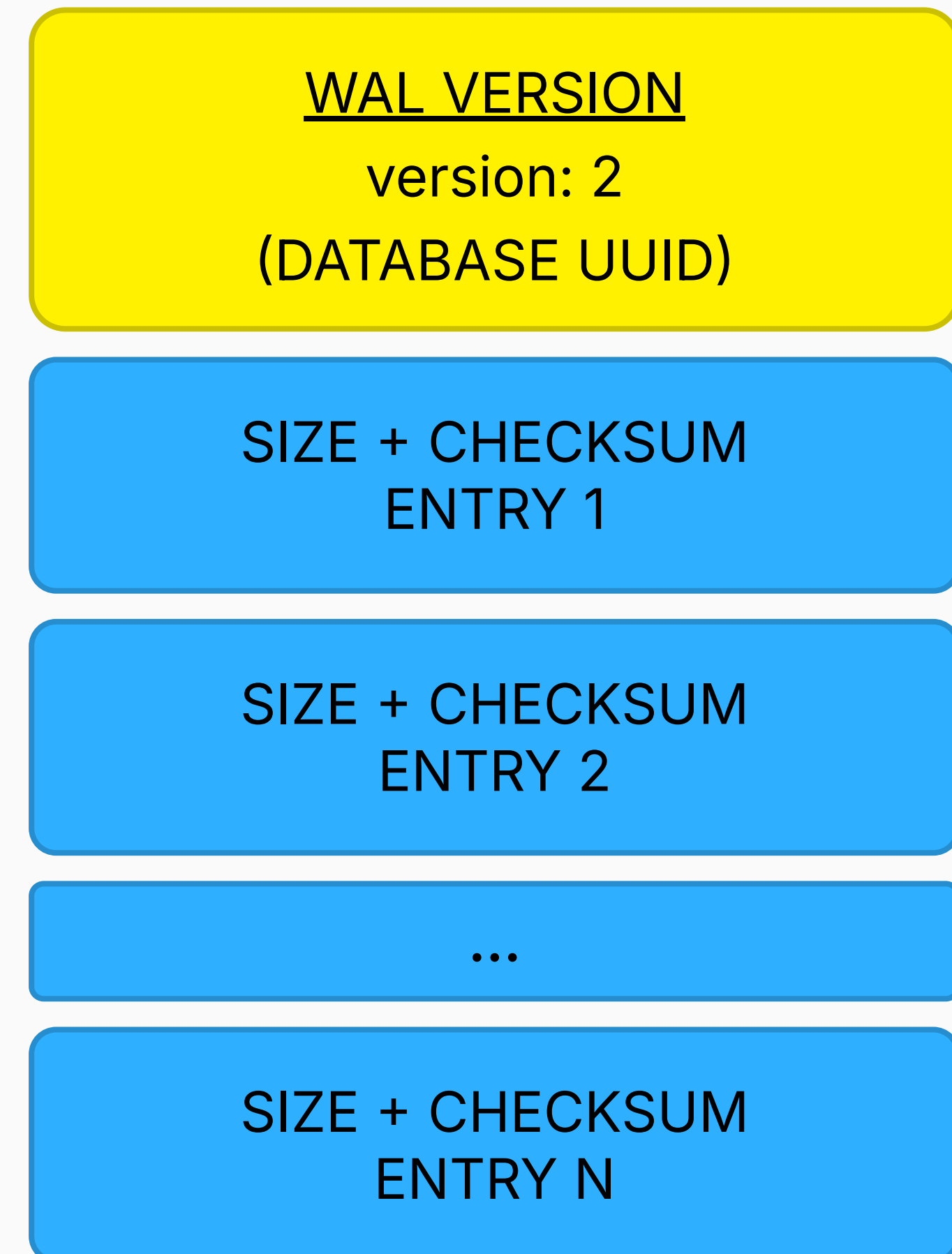


When is a WAL flushed?

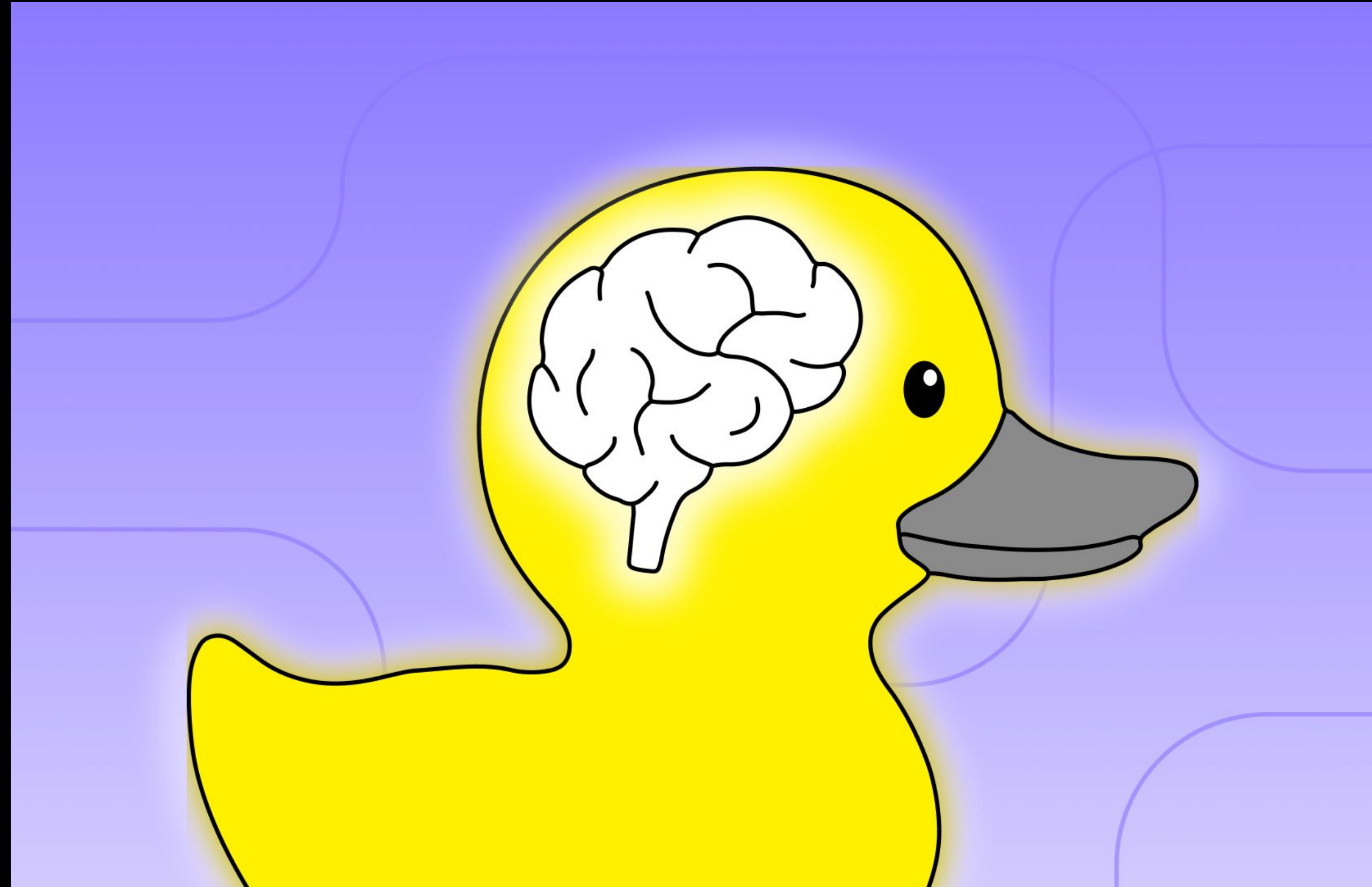


- CHECKPOINT;
- Automatic checkpoint
- Closing the database

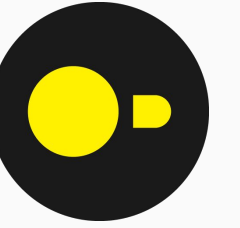
file.db.wal



**How Do We
Encrypt the WAL?**



The WAL is encrypted per entry



file.db.wal

WAL VERSION
version: 2
(DATABASE UUID)

SIZE + CHECKSUM
ENTRY 1

SIZE + CHECKSUM
ENTRY 2

SIZE + CHECKSUM
ENTRY N



encrypted.db.wal

WAL VERSION
version: 3
(DATABASE UUID)

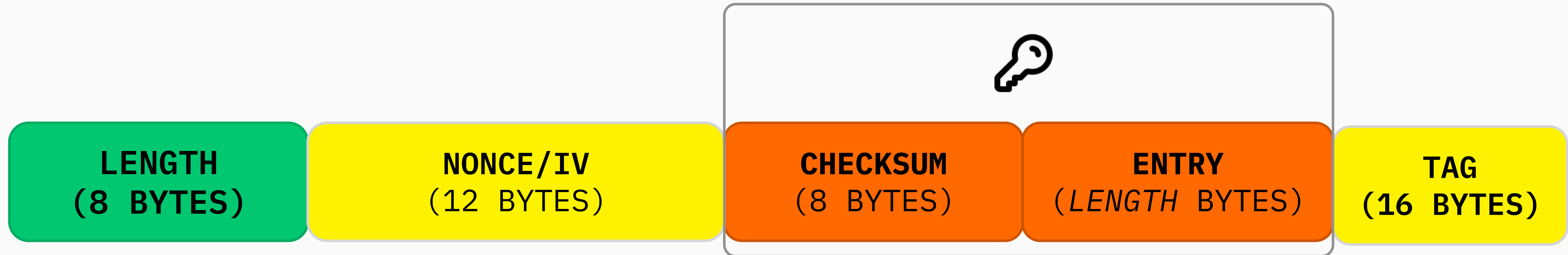
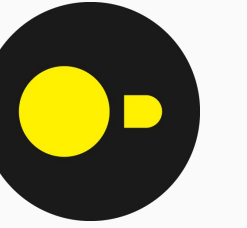
 SIZE + CHECKSUM
ENTRY 1

 SIZE + CHECKSUM
ENTRY 2

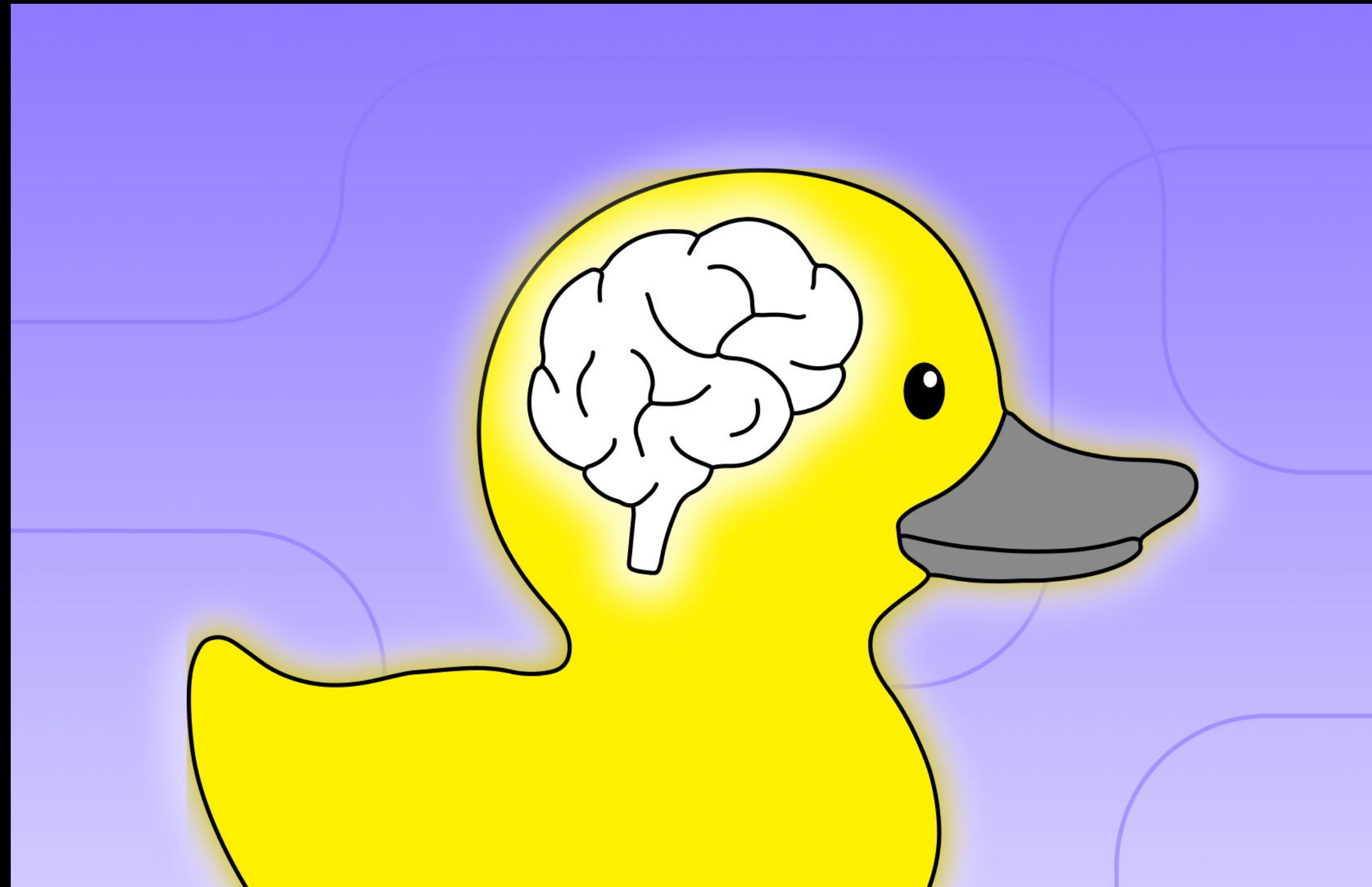


 SIZE + CHECKSUM
ENTRY N

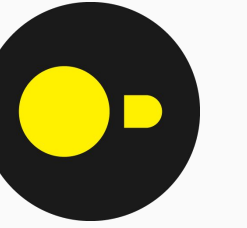
Encrypted WAL entries keep the length plaintext



Temporary File Encryption

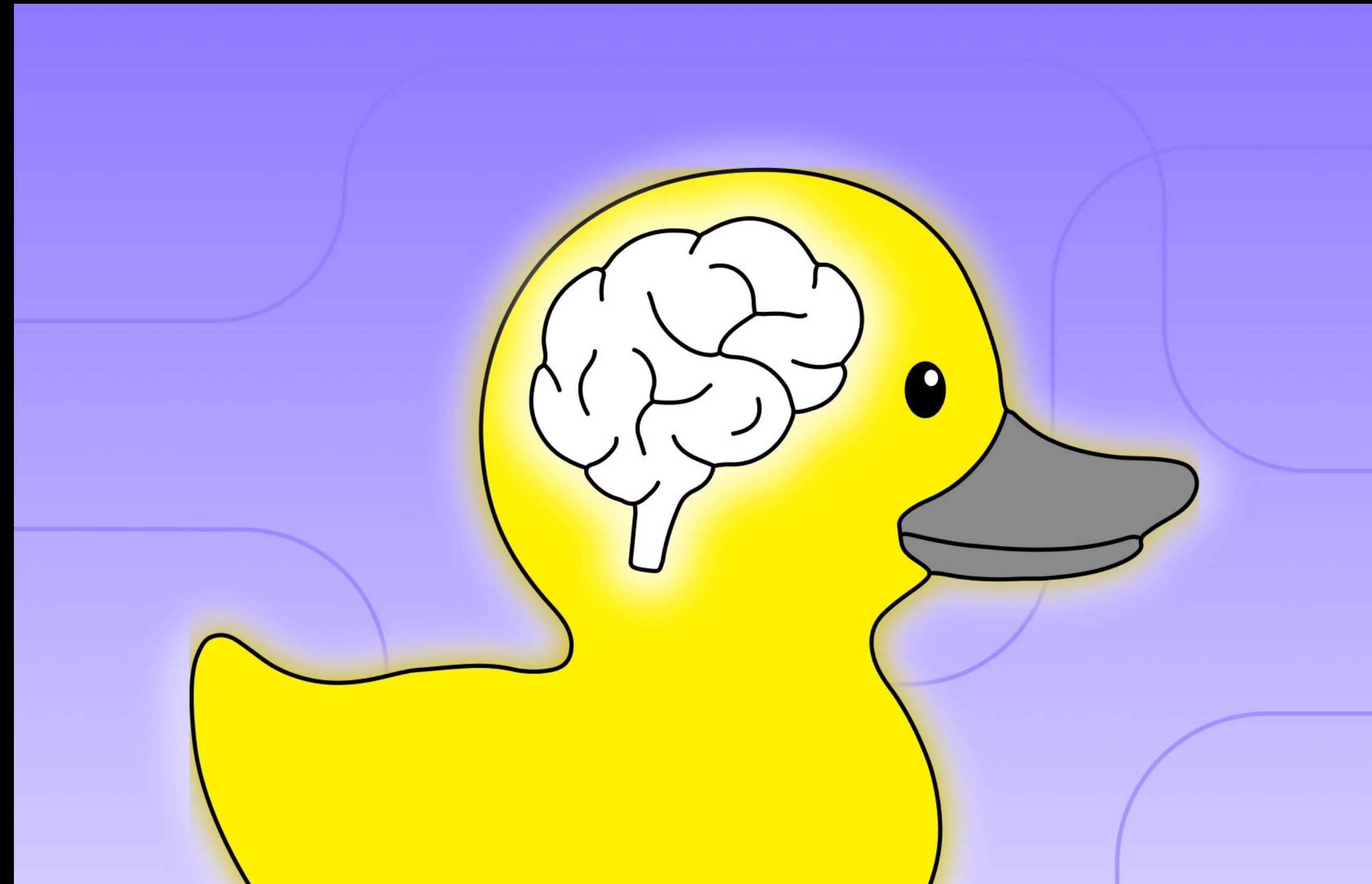


Temporary files

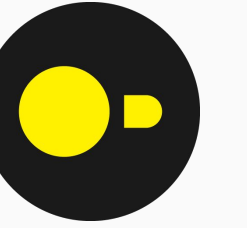


- Temporary Files are produced when data is **spilled to disk**
- They can be left behind in case of a **crash**
 - Or can be read during execution
- We also encrypt these files!

The Encryption API

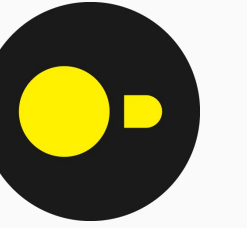


MbedTLS vs. OpenSSL



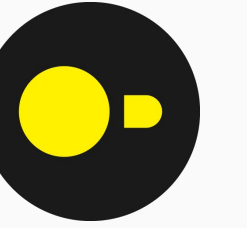
- By default, DuckDB has *no external dependencies*
- But how do we encrypt then?
 - Stripped MbedTLS library

MbedTLS vs. OpenSSL



- By default, DuckDB has *no external dependencies*
- But how do we encrypt then?
 - Stripped MbedTLS library
- But we want OpenSSL!

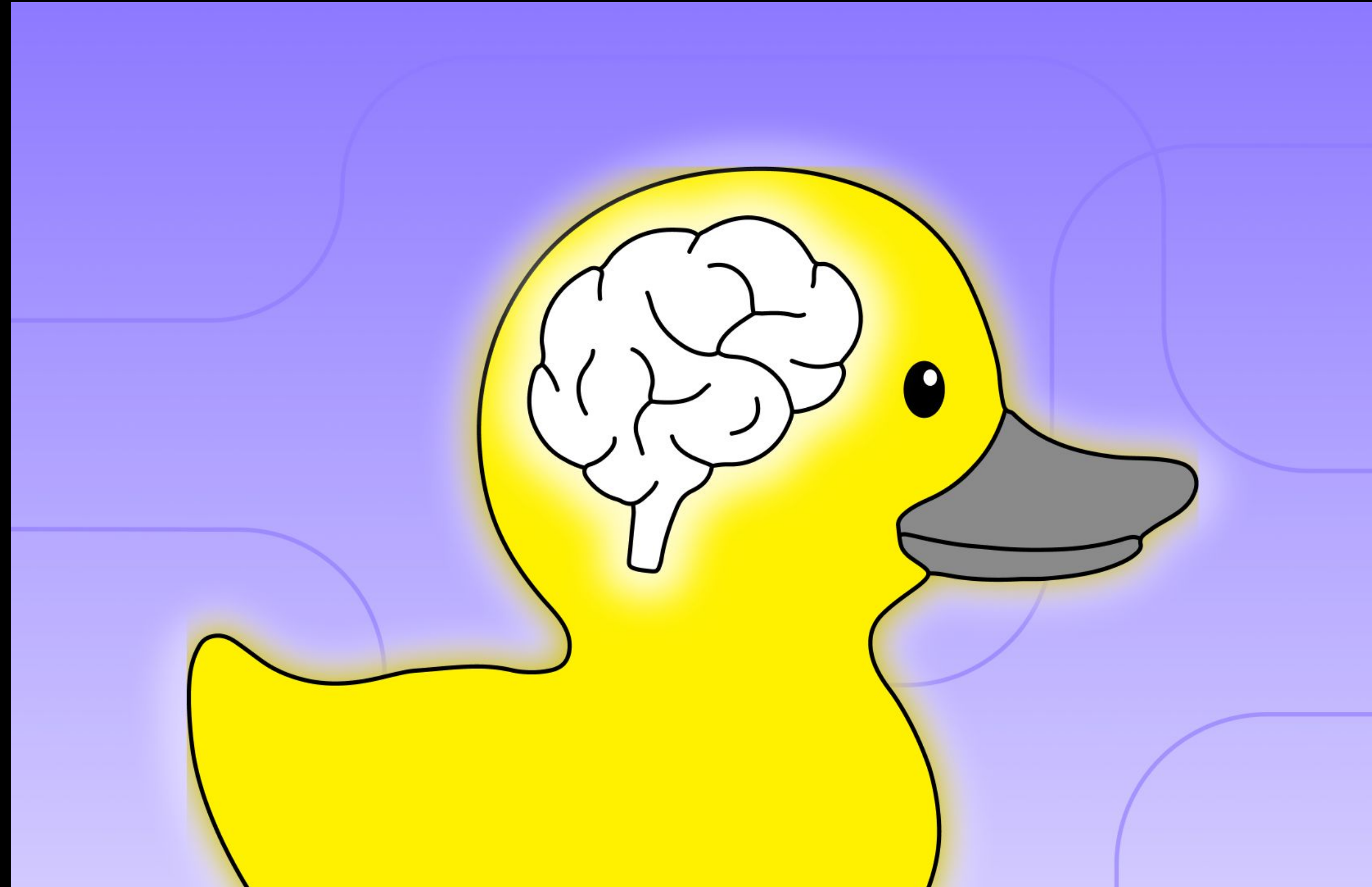
MbedTLS vs. OpenSSL



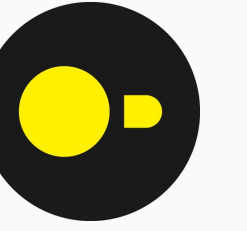
- The HTTPFS extension requires OpenSSL
 - We dynamically replace the mbedtls encryption API

	MbedTLS	OpenSSL
Reading	Secure	Secure
Writing	Unsafe (if not using dedicated hardware)	Secure
Performance overhead	Significant	Negligible

Conclusions



Takeaways



- DuckDB encrypts data-at-rest
- Encrypt the database file, along with WAL files and temporary files
- Encryption keys are managed securely
- Encryption incurs negligible overhead

Thank you!

