The Design and Implementation of

FUSIONDB

XML Prague
2019-02-08

Adam Retter

adam@evolvedbinary.com
@adamretter

EVOLVED BINARY

Why did we start in 2014?

- **Personal Concerns**
  - Open Source NXDs problems/limitations are not being addresed
  - Commercial NXDs are Expensive and not Open Source
  - New NoSQL (JSON) document db are out-innovating us
  - 10 Years invested in Open Source NXD, unhappy with progress
- **Commercial Concerns from Customers**
  - Help! Our Open Source NXD sometimes:
    - Crashes and Corrupts the database
    - Stops responding
    - Won't Scale with new servers/users

OS NXD - Known Issues ~2014

- **Reported by Users**
  - Stability - Responsiveness / Deadlocks
  - Operational Reliability - Backup / Corruption / Fail-over
  - Missing Feature - Key/Value Metadata for Documents
  - Missing Feature - Triple/Graph linking for inter/cross-document references
- **Recognised by Developers**
  - Correctness - Crash Recovery / Deadlock Avoidance / Deadlock Resolution / ACID
  - Performance - Reducing Contention / Avoiding System Maintenance mutex
  - Missing Features - Multi Model / Clustering

Hold My Beer...

I Gotta Fix This!

EVOLVED BINARY

Can we fix an existing NXD?

- **Project Health?**
  - Issues - Rate of decay, i.e. Open vs Closed over time
  - Attracts new contributors?
  - Attracts new and varied users?
  - How do contributors pay their bills?
- **Contributor Constraints?**
  - How long to get PRs reviewed?
  - Open to radical changes? Incremental vs Big-bang?
  - Other developers with time/knowledge to review PRs
- **License**
  - Business friendly? CLA?
- **Reputation - Perceived or otherwise**

Time to build something new

- **Project "Granite"**
  - Research and Development
  - Primary Focus on Correctness and Stability
    - Never become unresponsive
    - Never crash
    - Never lose data or corrupt the database
- **Should become Open Source**
  - Should be appealing to Commercial enterprises
  - Open Source license choice(s) vs Revenue opportunities
- **Don't reinvent wheels!**
  - Reuse - Faster time to market
  - Developers know eXist-db... Fork it!

First... Replace eXist-db's Storage Engine

- **Why?**
  - We don't trust it's correctness
  - Old and Creaky? - (dbXML ~2001)
    - Improved with caching and journaling
  - Not Scalable - single-threaded read/write
  - Classic B+ Tree
  - Why not fix it?
    - Newer/better algorthms exist - B-link Tree, Bw Tree, etc.
    - We want a giant-leap, not an incremental improvement

How does a NXD Store an XML Document Anyway?

...Shredding!

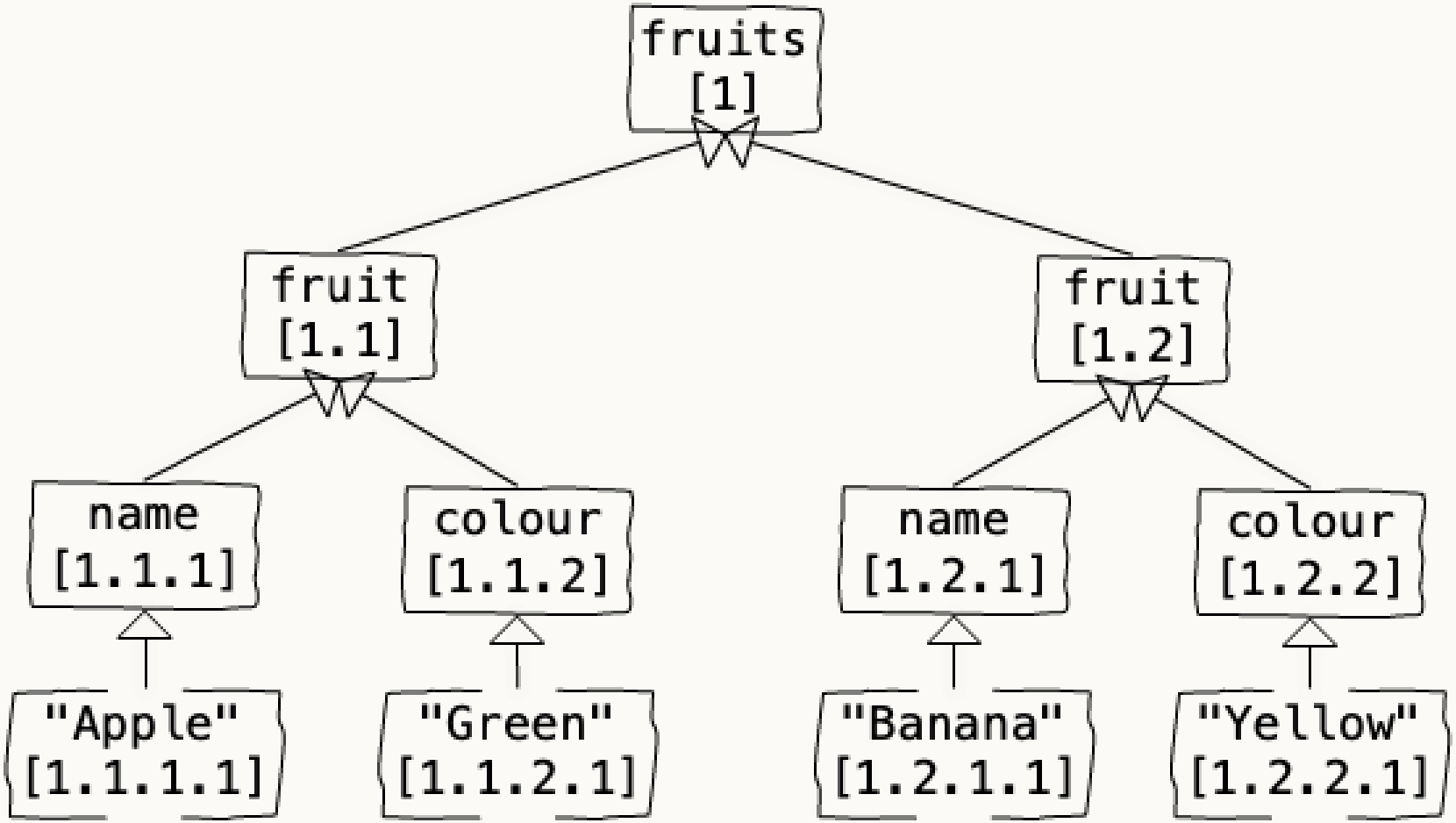
EVOLVED BINARY

Given some very simple XML - *fruits.xml*

```
1  <fruits>
2      <fruit>
3          <name>Apple</name>
4          <colour>Green</colour>
5      </fruit>
6      <fruit>
7          <name>Banana</name>
8          <colour>Yellow</colour>
9      </fruit>
10 </fruits>
```

1. Number the tree (DLN)

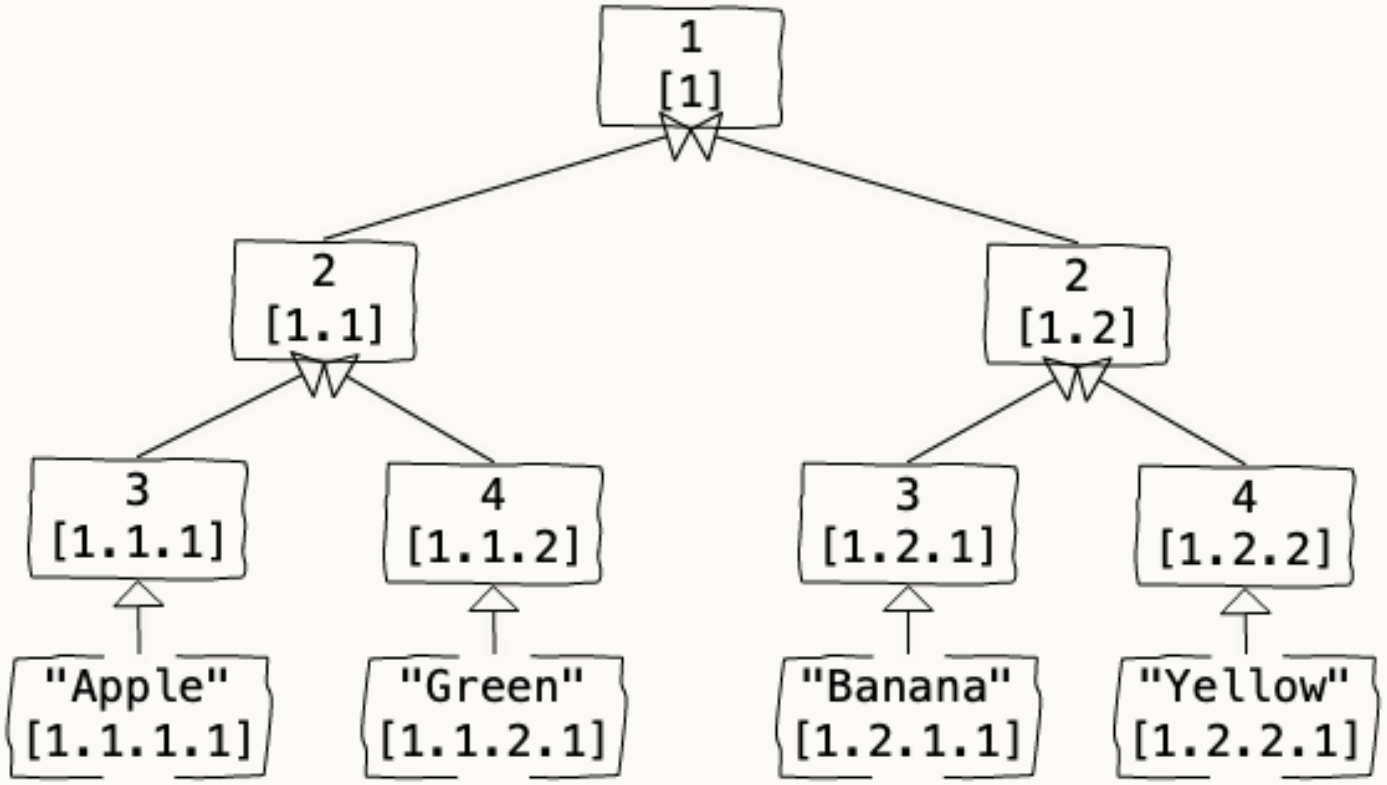
fruits.xml: docId=6



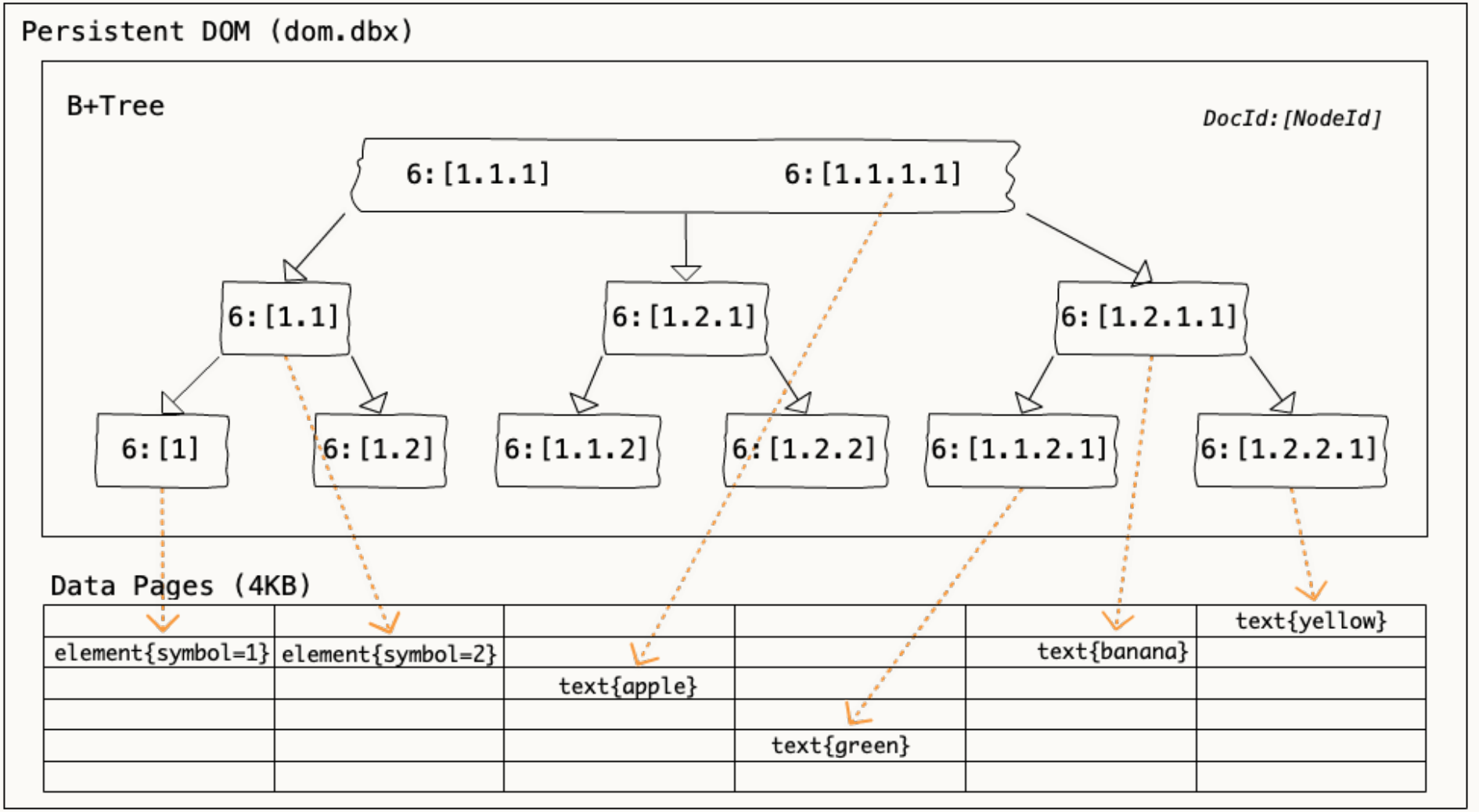
2. Extract Symbols

Symbol Table

ID	Symbol
1	fruits
2	fruit
3	name
4	colour

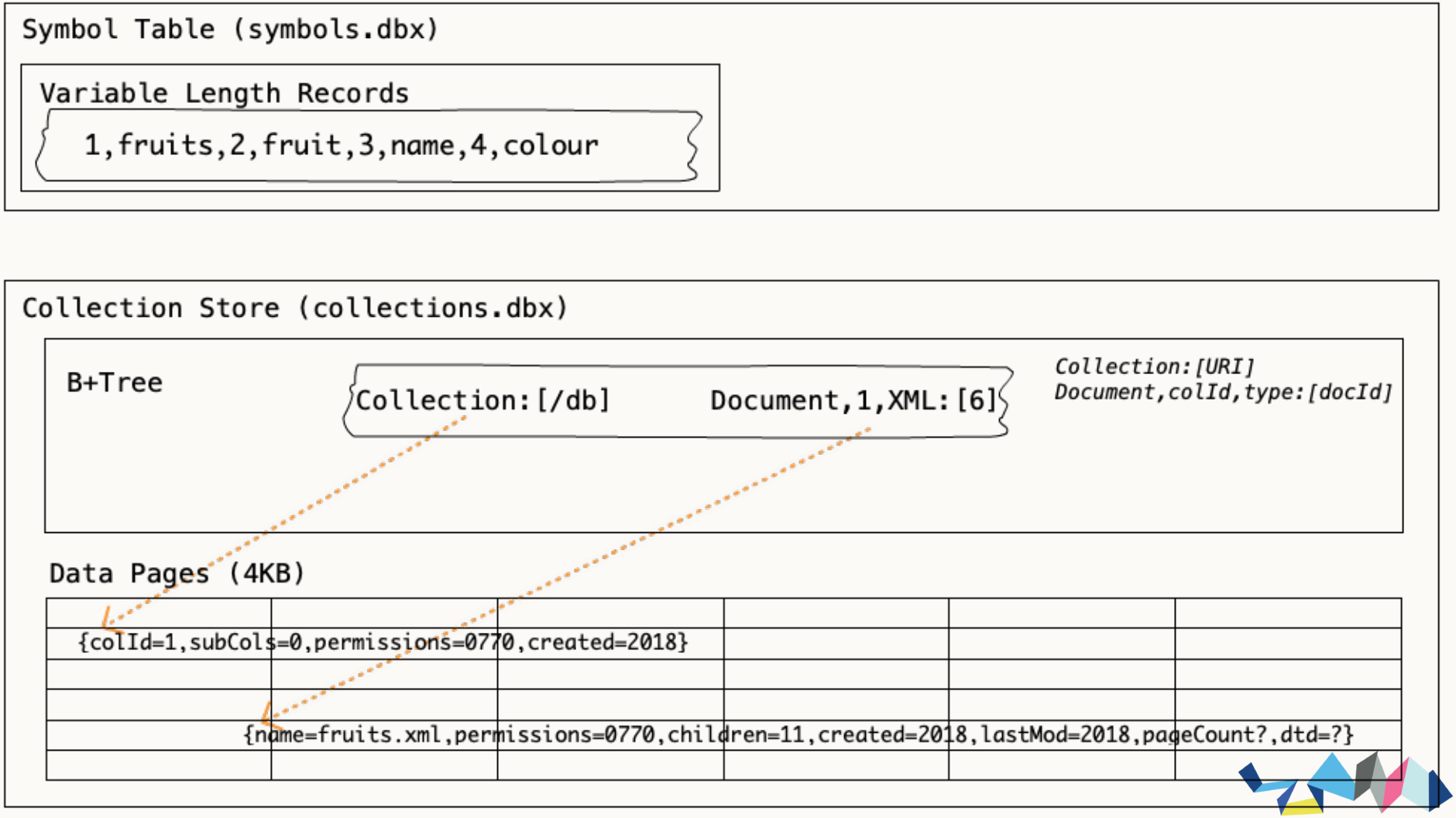


3. Store DOM



EVOLVED BINARY

4. Store Symbols + Collection Entry



EVOLVED BINARY

New Storage Engine

- **We won't develop our own!**
  - It's hard (to get correct)!
  - Other well-resourced projects available for reuse
  - We would rather focus on the larger DBMS
- **Choosing a suitable engine**
  - Other Java Database B+ Trees are unsuitable
    - Examined - Apache Derby, H2, HSQLDB, and Neo4j
  - Few Open Source pure Storage Engines in Java
    - Discounted MapDB - known issues
  - Identified 3 possibilities:
    - LMDB - B-Tree written in C
    - ForestDB - HB+-Trie written in C++11
    - RocksDB - LSM written in C++14

Why we Adopted RocksDB

- **Fork of Google's LevelDB**
  - Performance Improvements for concurrency and I/O
  - Optimised for SSDs
- **Large Open Source community with commercial interests**
  - e.g. Facebook, AirBnb, LinkedIn, NetFlix, Uber, Yahoo, etc.
- **Rich feature set**
  - LSM-tree (Log Structured Merge Tree)
  - MVCC (Multi-Version Concurrency Control)
  - ACID
    - Built-in **A**tomicity and **D**urability
    - Offers primitives for building **C**onsistency and **I**solation
  - Column Families

How do we store XML into RocksDB?

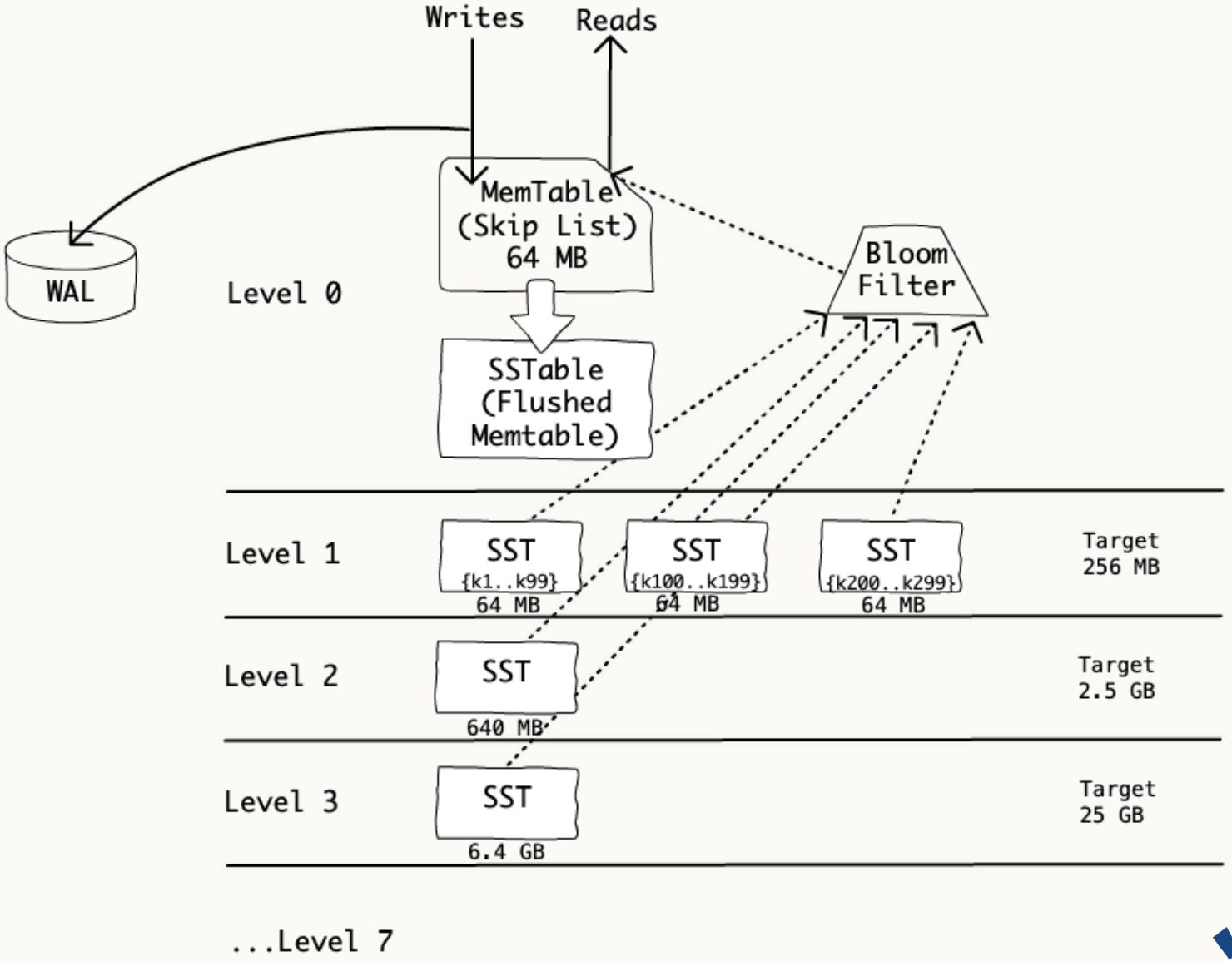
...Mooor Shredding!

EVOLVED BINARY

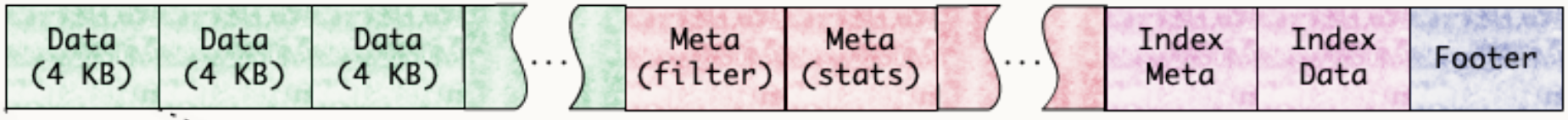
Storing XML in RocksDB

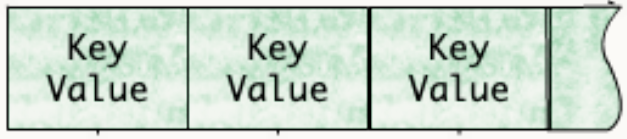
- **We still:**
  - Number the tree with DLN
  - Extract Symbols
  - Shred each node into a key/value pair
- **We do NOT use eXist's B+Tree or Variable Record Store**
- **Instead, we use RocksDB Column Families**
  - Each is an LSM-tree. Share a WAL
  - For each component
    - Persisent DOM Column Family - XML_DOM_STORE
    - Symbol Table Column Families - SYMBOL_STORE, NAME_INDEX, NAME_ID_INDEX
    - Collection Store Column Family - COLLECTION_STORE

RocksDB's LSM Tree



XML in RocksDB's SSTable File






```
Key         - Value
6,1         - element{symbol=1}
6,1.1       - element{symbol=2}
6,1.1.1     - element{symbol=3}
6,1.1.1.1   - text{Apple}
6,1.1.2     - element{symbol=4}
6,1.1.2.1   - text{Green}
6,1.2       - element{symbol=2}
6,1.2.1     - element{symbol=3}
6,1.2.1.1   - text{Banana}
6,1.2.2     - element{symbol=4}
6,1.2.2.1   - text{Yellow}
```

EVOLVED BINARY

ACID Transactions

- **eXist-db lacks strong transaction semantics**
  - Txn - Log commit/abort just for Crash Recovery
  - Mostly just the Durability of ACID
  - Isolation level: ~Read Uncommitted
- **Our Transactions**
  - RocksDB ensures **A**tomicity and **D**urability
  - We add **C**onsistency and **I**solation
  - Begin Transaction creates a db Snapshot
  - Each Transaction has an in-memory Write Batch
    - Write - only to in-memory Write Batch
    - Read - try in-memory write batch, fallback to Snapshot
  - Isolation level: >= Snapshot Isolation

Transactions for Users

- **Each public API call is a transaction**
  - e.g. REST / WebDAV / RESTXQ / XML-RPC / XML:DB
  - auto-abort on exception
  - auto-commit when the call returns data
- **Each XQuery is a transaction**
  - auto-abort if the query raises an error
  - auto-commit when the query completes
  - Begin Transaction creates a db Snapshot
  - XQuery 3.0 try/catch
    - try - begins a new sub-transaction
    - catch - auto-abort, the operations in the try body are undone
    - Sub-transactions can be nested, just like try/catch

Other new features include...

- **Key/Value Model**
  - Metadata for Documents and Collections
  - Searchable from XQuery
- **Online Backup**
  - Lock free
  - Checkpoint Backup
  - Full Document Export
- **UUID Locators**
  - Persist across backups and nodes
- **BLOB Store**
  - Deduplication

Reflections

- **Many Changes Upstreamed**
  - eXist-db - Locking, BLOB Store, Concurrency, etc.
  - RocksDB - further Java APIs and improved JNI performance
- **With hindsight...**
  - Wouldn't fork eXist-db...
    - Too much time spent discovering and fixing bugs
    - Start with a green field, add eXist-db compatible APIs
  - Much more work than anticipated
- **Today, new storage engines**
  - FoundationDB / BadgerDB / FASTER

On the roadmap...

- Benchmarking and Performance
- JSON Native
- Virtualised Collections
- Distributed Cluster
- Graph Model
- XQuery compilation to native CPU/GPU code

EVOLVED BINARY