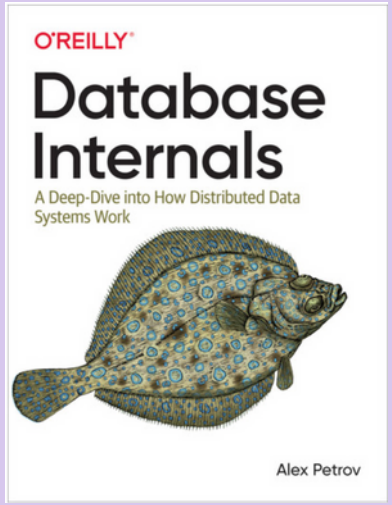
CHAPTER 6
B-TREE VARIANTS

DB INTERNALS MEETUP

VERITY CHU

2 MAR, 2022



B-Tree variants

common:

tree structure

balancing (split / merge)

lookup / delete algorithms

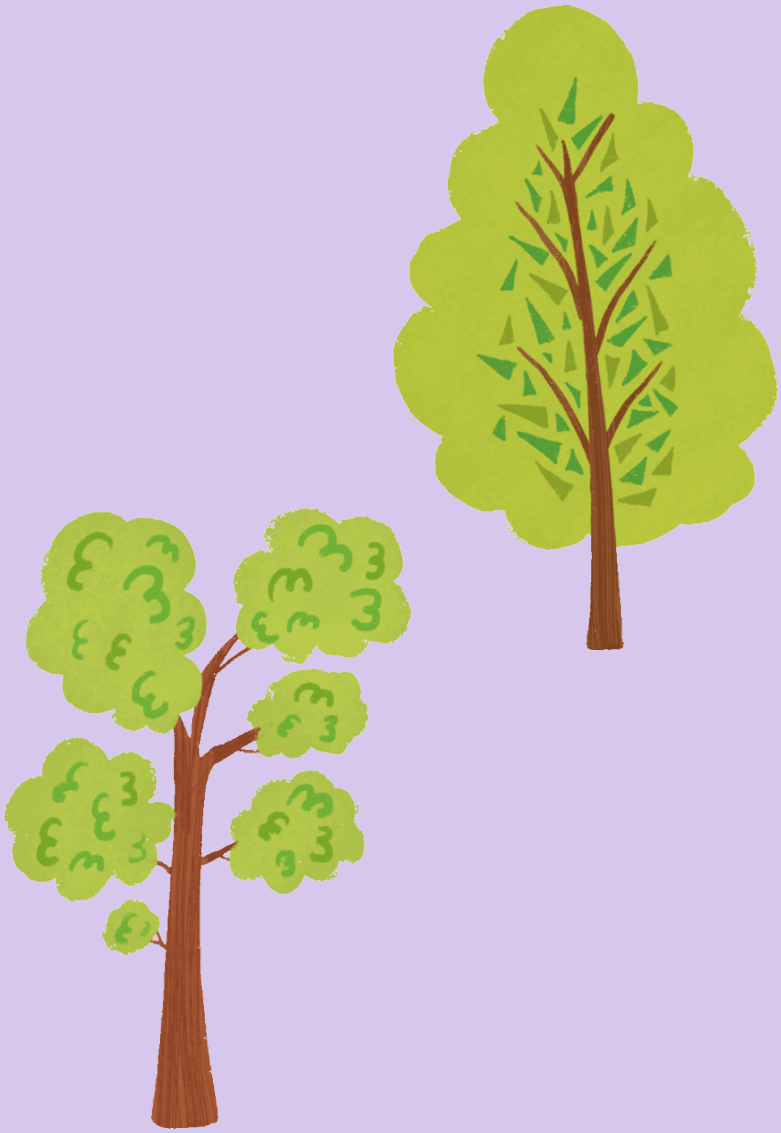
vary:

concurrency

on-disk page representation

sibling nodes links

maintenance processes



B-Tree variants

- *Copy-on-write B-Trees*
- *Lazy B-Trees*
- *FD-Trees*
- *Bw-Trees*
- *Cache-oblivious B-Trees*

TODAY'S SUMMARY

- COPY-ON-WRITE B-TREES
- LAZY B-TREES
- FD-TREES

Copy-on-Write B-Tree

Concurrent operations...

guarantee data integrity

Latching?

Copy-on-write!

Copy-on-Write B-Tree

Concurrent operations...

> whenever the page is about to be modified

1. *Its contents are copied*
2. *The copied page is modified*
3. *A parallel tree hierarchy is created*

Copy-on-Write B-Tree

Concurrent operations...
> whenever the page is about to be modified

***Readers**: can access the old tree versions concurrently.*

***Writers**: accessing modified pages have to wait until preceding write operations are complete.*

Copy-on-Write B-Tree

***Readers**: can access the old tree versions concurrently.*
***Writers**: accessing modified pages have to wait until preceding write operations are complete.*

After the new page hierarchy is created, the pointer to the topmost page is atomically updated.

Copy-on-Write B-Tree

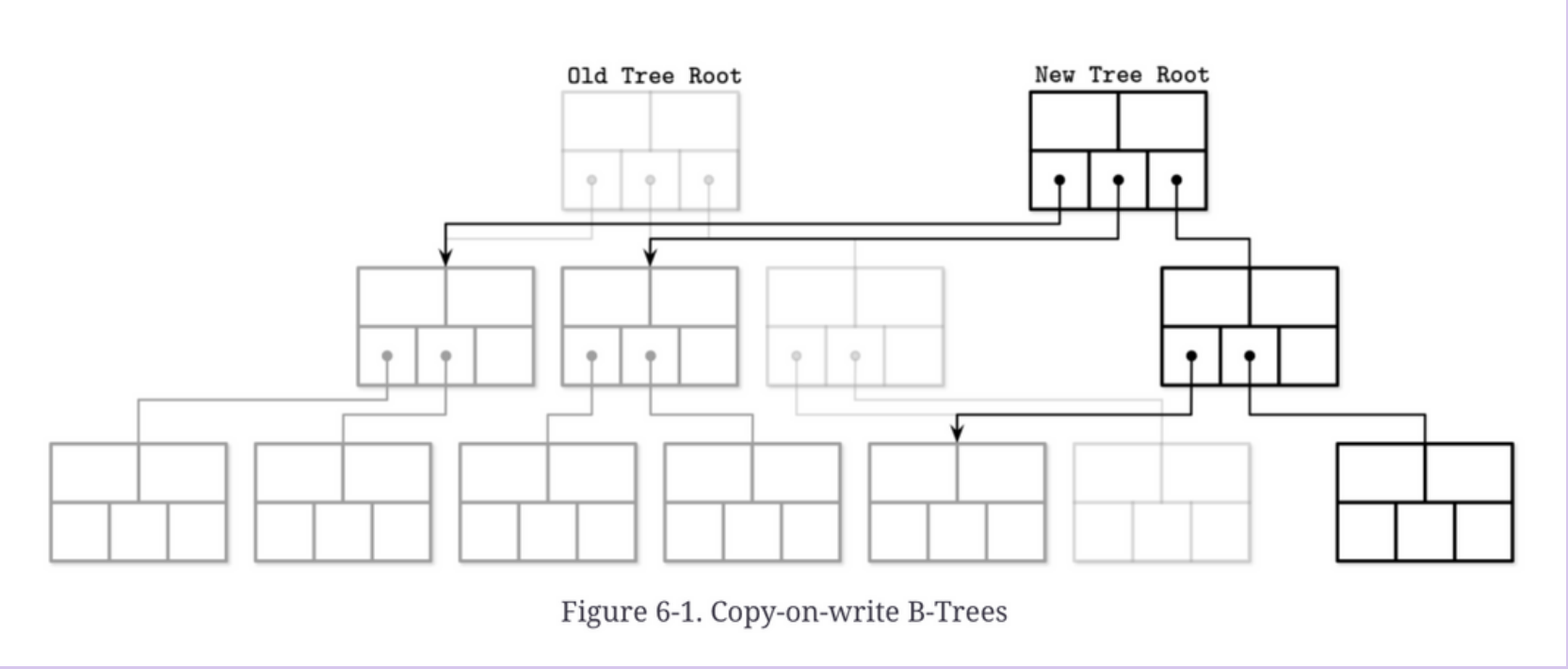


Figure 6-1. Copy-on-write B-Trees

After the new page hierarchy is created, the pointer to the topmost page is atomically updated.

Copy-on-Write B-Tree

Downside:

+ more space: retain old page

+ processor time: copy entire page content

Copy-on-Write B-Tree

Since B-Tree are generally shallow

Simplicity & advantages still outweigh!

Copy-on-Write B-Tree

readers: require no synchronisation

> written pages are immutable and can be accessed without additional latching.

writers: performed against copied pages

> readers do not block writers.

Copy-on-Write B-Tree

> No operation can observe a page in an incomplete state

> system crash cannot leave pages in a corrupted state

Copy-on-Write B-Tree: LMDB

Lightning Memory-Mapped Database (LMDB)

a key-value store used by the OpenLDAP project

https://github.com/LMDB/lmdb

Copy-on-Write B-Tree: LMDB

Implemented as a Single-level data store

> read and write operations are satisfied directly through the memory map, without additional application-level caching in between.

> pages require no additional materialisation

> reads can be served directly from the memory map

Copy-on-Write B-Tree: LMDB

During the update, every branch node on the path from the root to the target leaf is copied and potentially modified.

Rest of the nodes remain intact.

Copy-on-Write B-Tree: LMDB

root node:
LMDB holds only 2 versions

1: latest version

2: the one new changes going to be committed

Copy-on-Write B-Tree: LMDB

Sibling nodes:

LMDB's append-only design

> Does NOT use sibling pointers and has to ascend back to the parent node during sequential scans.

Copy-on-Write B-Tree: LMDB

> Inherently multiversioned (MVCC).

> Readers can run without any locks

Lazy B-Trees

reduce costs of updating the B-Tree and use more lightweight, concurrency- and update-friendly in-memory structures to:
buffer updates and propagate them with a delay.

Lazy B-Trees: WiredTiger

By MongoDB
Its row store B-Tree implementation uses different formats for in-memory and on-disk pages

Before in-memory pages are persisted, they have to go through the **reconciliation** process.

https://github.com/wiredtiger

Lazy B-Trees: WiredTiger

***Clean* page:** consists of just an index

Update buffer: updates are first saved into this.

Lazy B-Trees: WiredTiger

***Clean* page:** consists of just an index

Update buffer: updates are first saved into this.

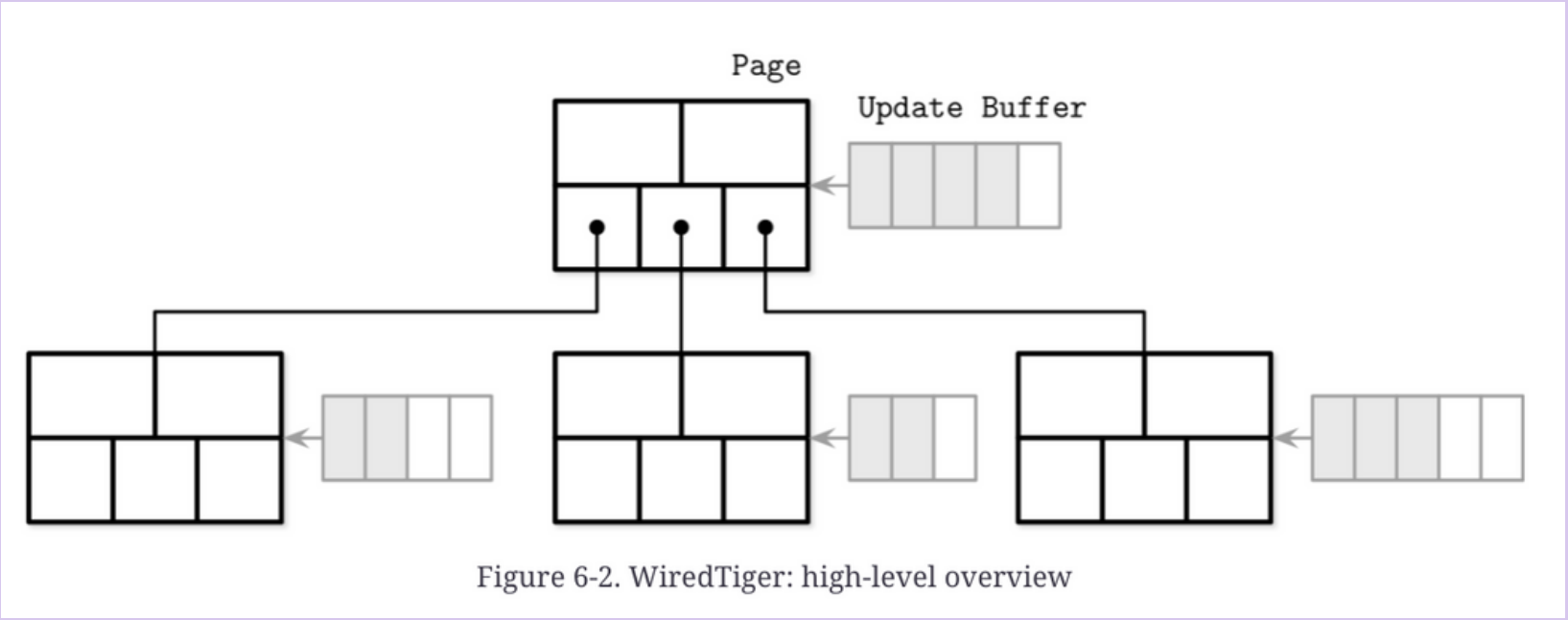


Figure 6-2. WiredTiger: high-level overview

Lazy B-Trees: WiredTiger

Update buffer

accessed during **reads**:
their contents are **merged** with the original on-disk page contents to return the most recent data.

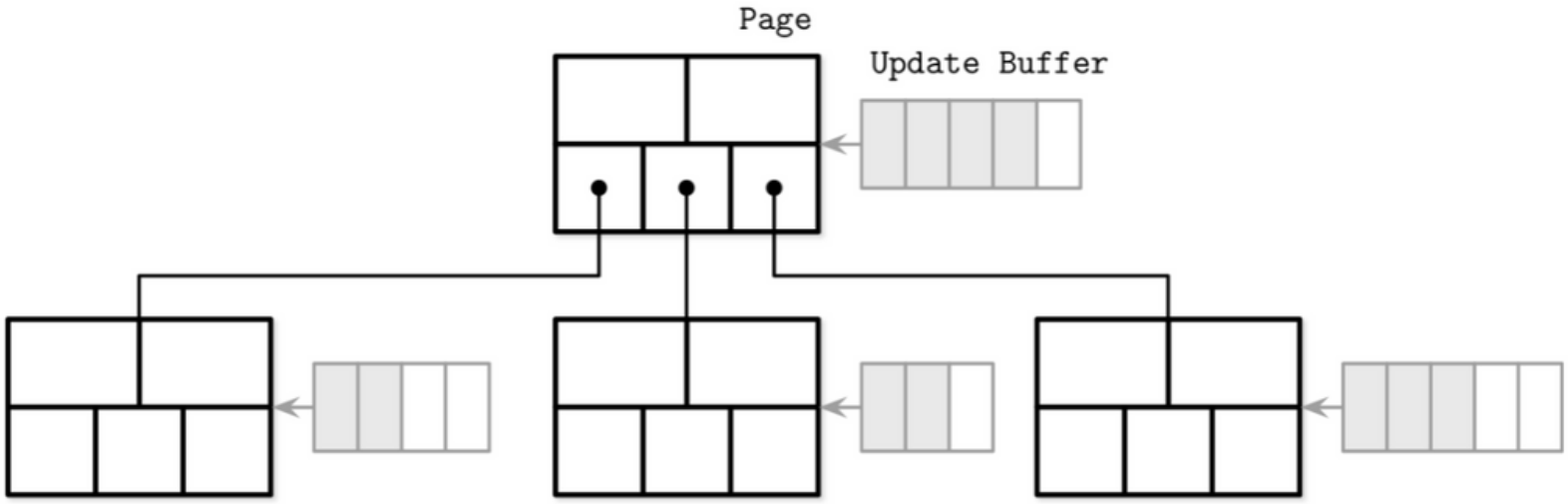


Figure 6-2. WiredTiger: high-level overview

Lazy B-Trees: WiredTiger

Update buffer

When the page is **flushed**:
update buffer contents are **reconciled** with page contents and persisted on disk, **overwriting** the original page.

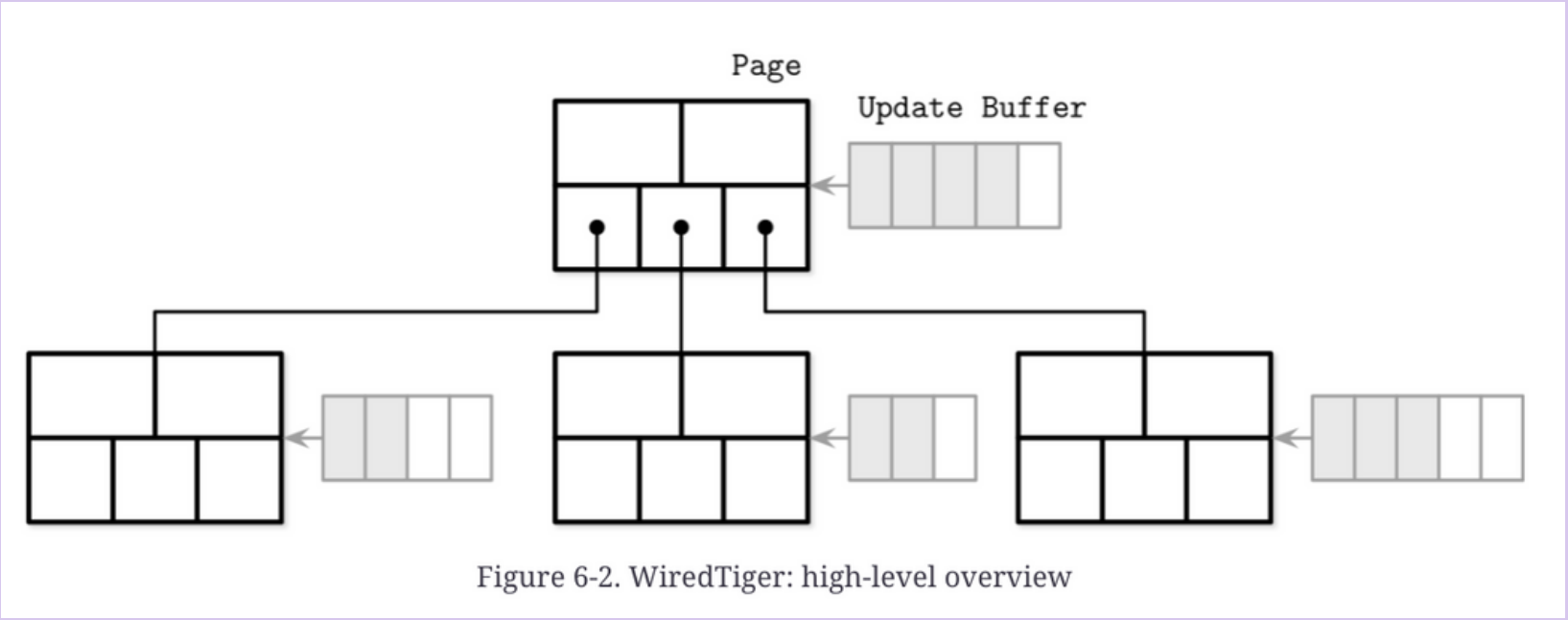


Figure 6-2. WiredTiger: high-level overview

Lazy B-Trees: WiredTiger

Update buffer

implemented using skiplists, which have a complexity similar to search trees but have a **better concurrency** profile.

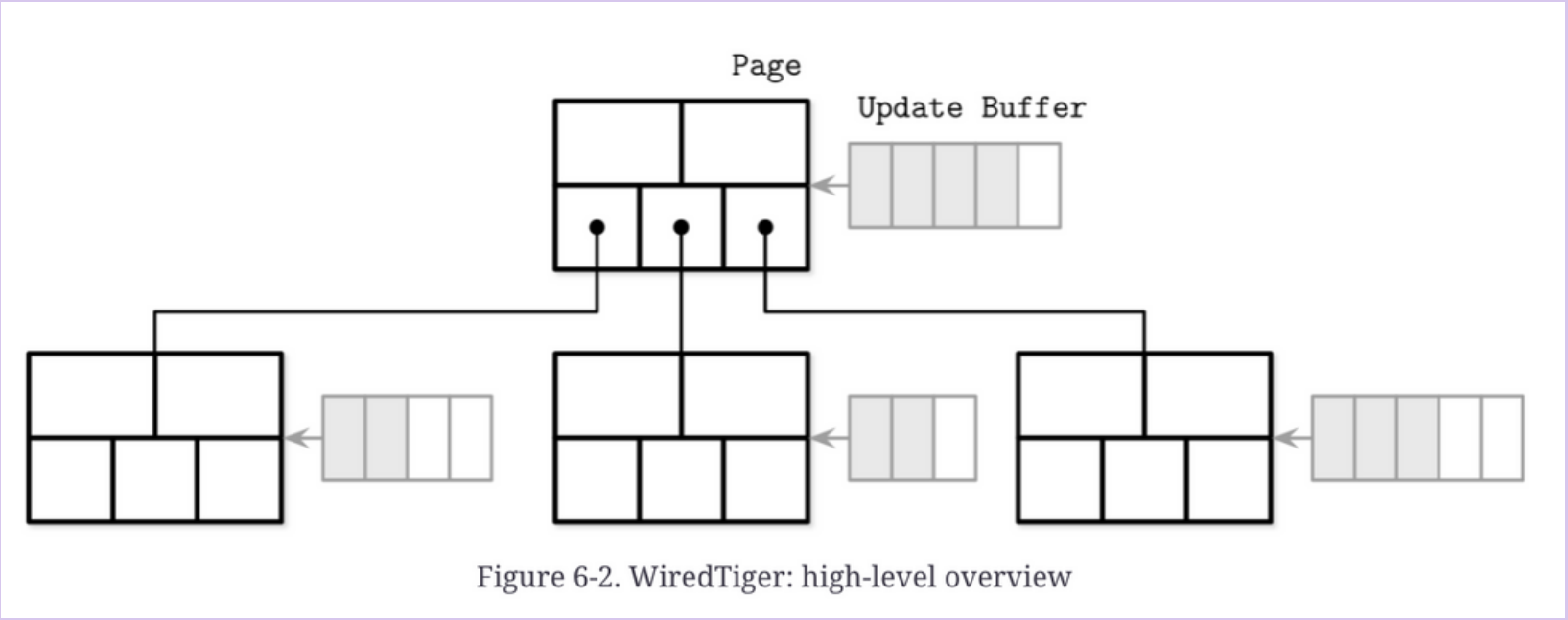


Figure 6-2. WiredTiger: high-level overview

Lazy B-Trees: WiredTiger

Both **clean** and **dirty pages** in WiredTiger have **in-memory versions**, and reference a **base image on disk**

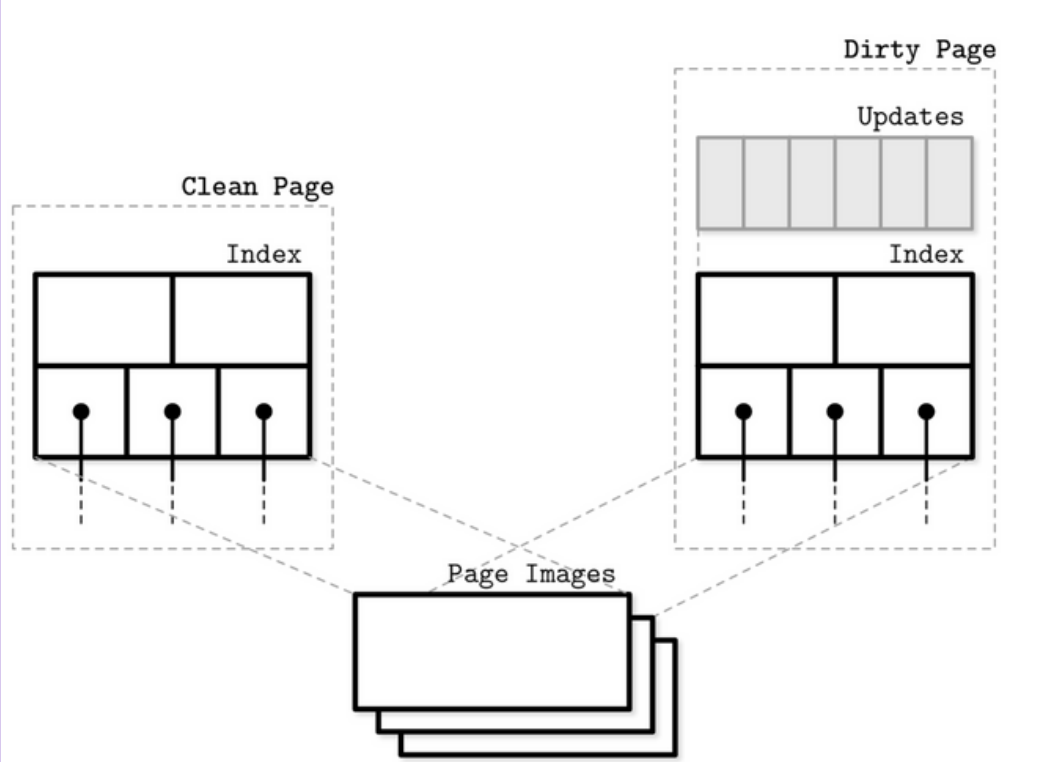


Figure 6-3. WiredTiger pages

Lazy B-Trees: WiredTiger

Advantages:
> page updates and structural modifications (splits and merges) are performed by the background thread,
> read/write processes do not have to wait for them to complete.

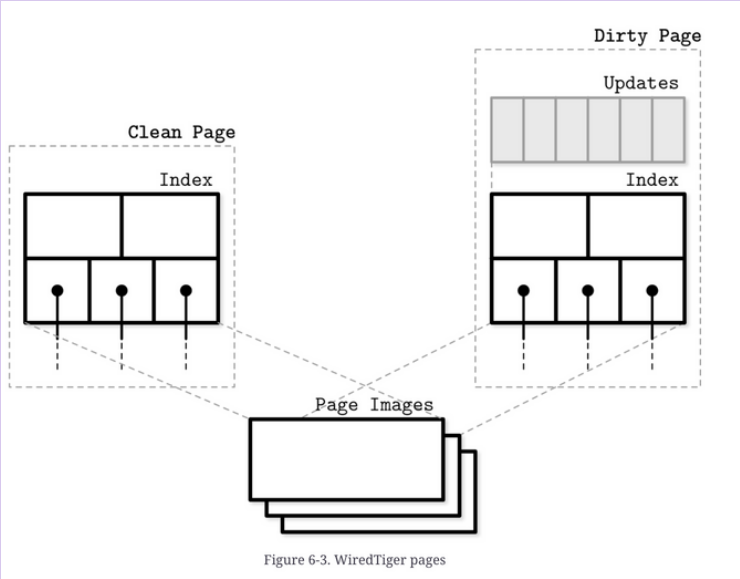


Figure 6-3. WiredTiger pages

Lazy-Adaptive Tree (LA-Tree)

Why not group nodes into subtrees and attach an update buffer for batching operations *to each subtree?*

Lazy-Adaptive Tree (LA-Tree)

Update buffers : track all operations performed against the subtree top node and its descendants, **recursively**

cascaded buffers

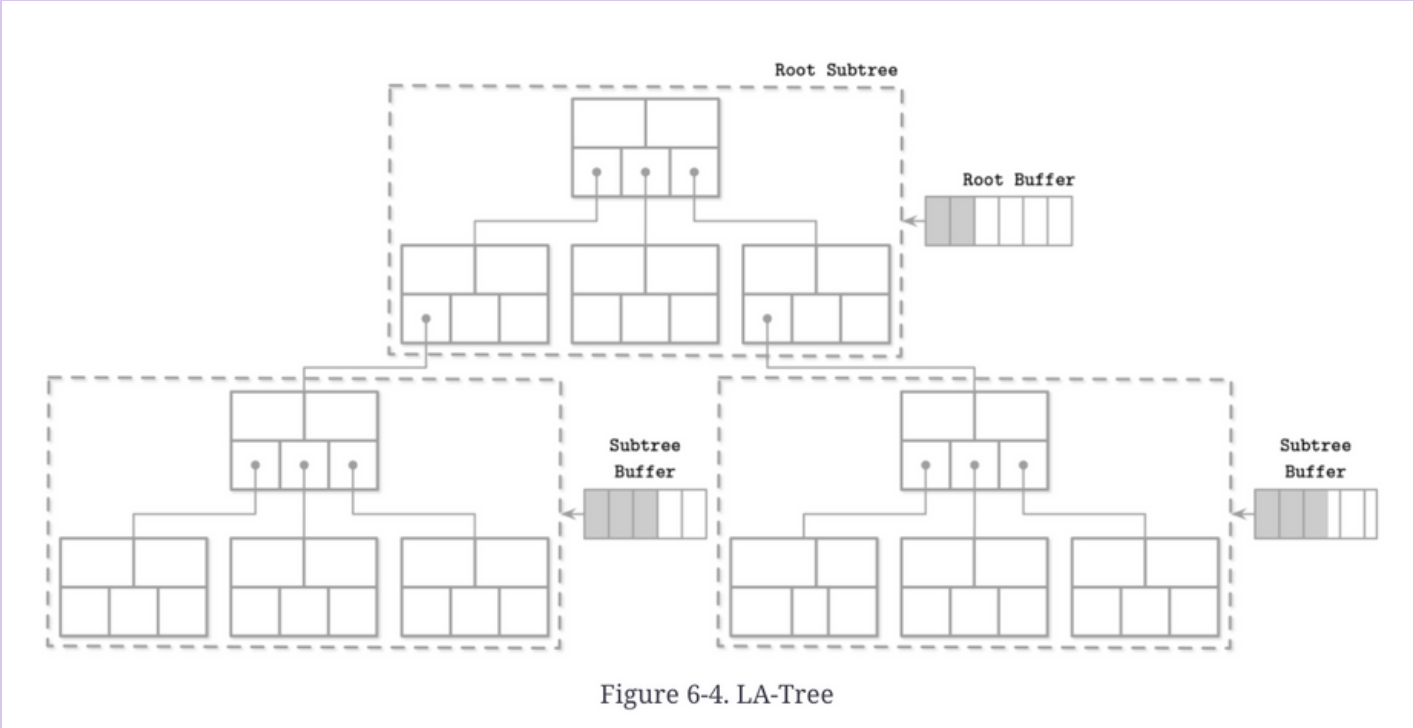


Figure 6-4. LA-Tree