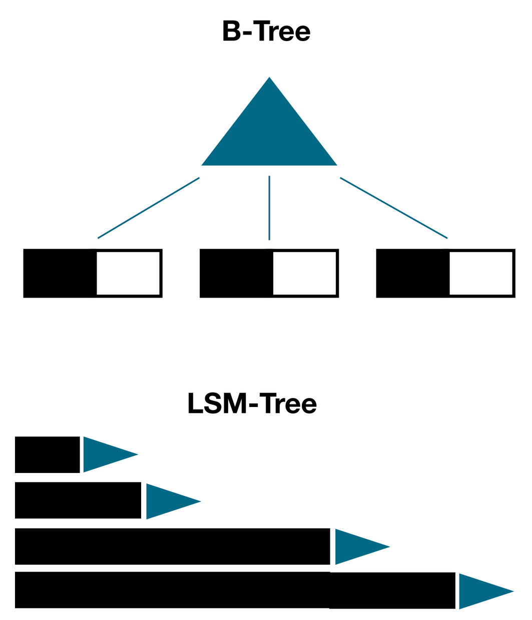
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

LSM-Trees and B-Trees: The Best of Both Worlds

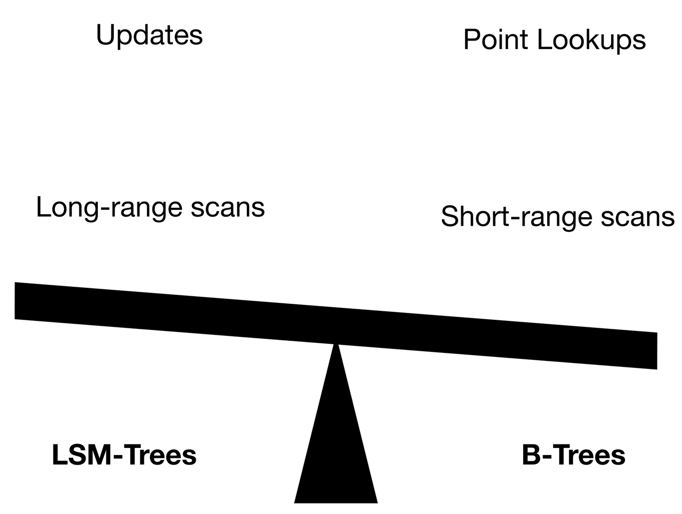
J. Lennon, V. Jain, H. Gupta
Advisor: Stratos Idreos

Key-Value Stores Today Are Suboptimal for Dynamic Workloads

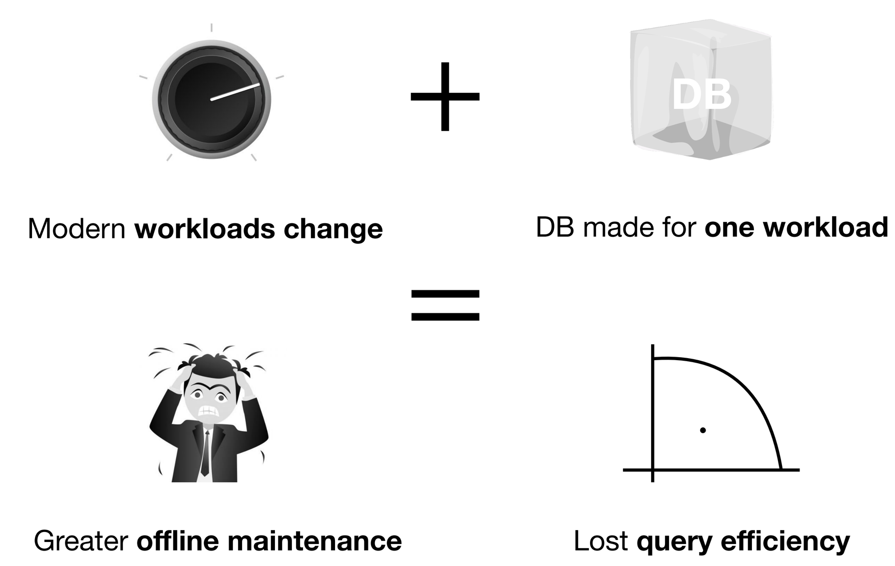
NoSQL key-value stores are widely popular today.



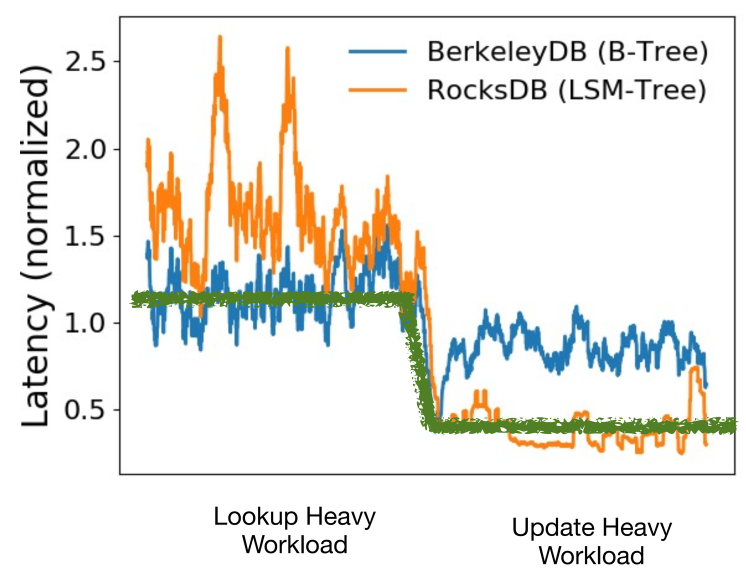
LSM-Tree and B-Tree data structures commonly back key-value stores.



An optimal data structure design is determined by the **specific workload distribution.**



Key-value stores optimized for one fixed workload are **suboptimal** for dynamic modern applications.



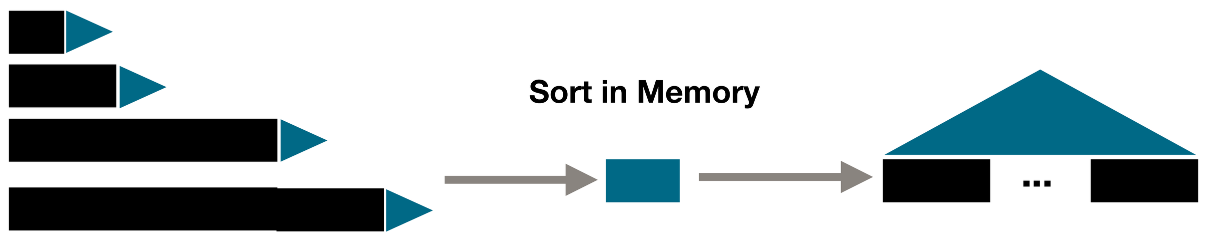
Our goal is to achieve the best latency on changing workloads with **on the fly transitions.**

The Solution: Transitions

Memory
Persistent Storage

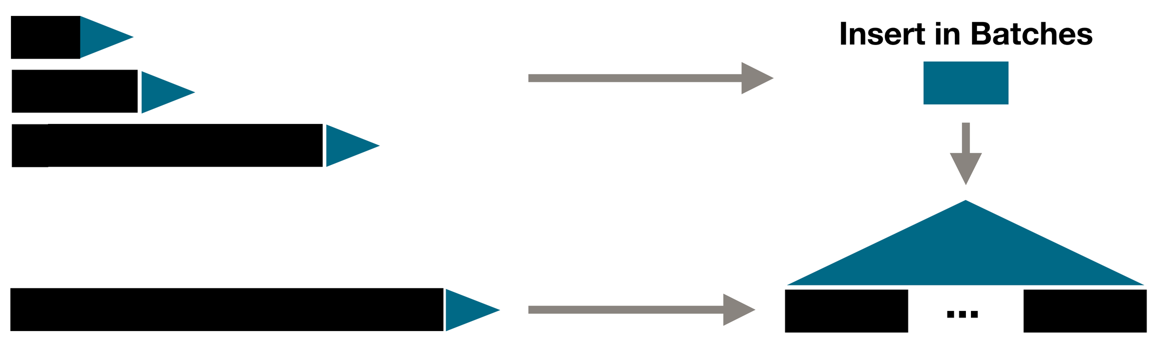
LSM-Tree -> B-Tree

Repeatedly remove the k blocks of entries with lowest keys from the LSM-Tree and append them to the end of the B-Tree leaf level.



OR

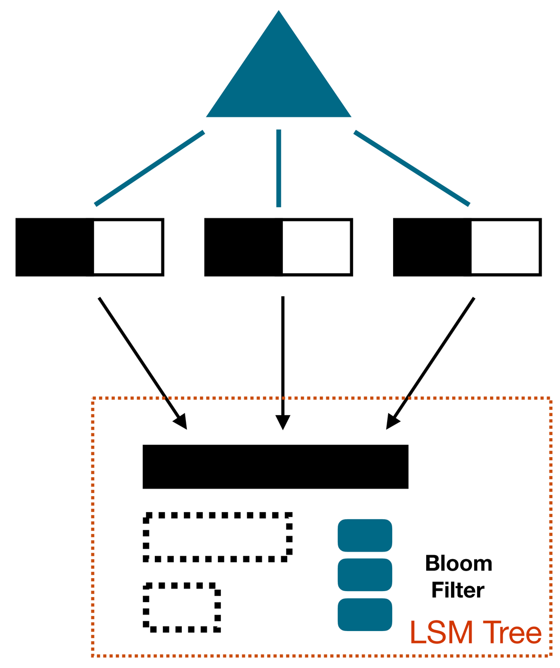
Convert the lowest level of the LSM-Tree into a B-Tree, avoiding disk IO. Then repeatedly insert batches of k blocks of entries with the lowest keys into the B-Tree.



B-Tree -> LSM-Tree

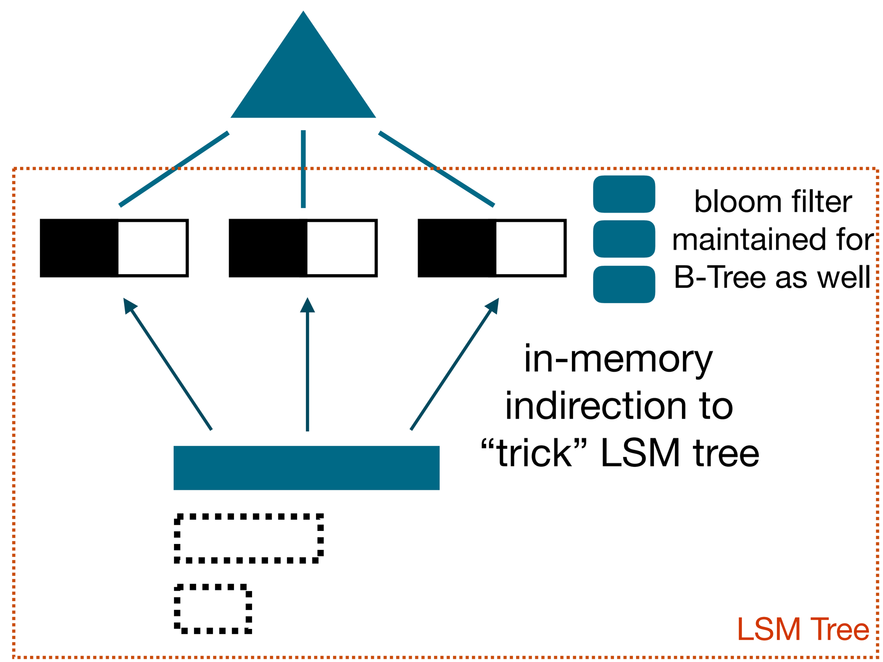
Similar underlying structure: **indexes in memory**, **data on disk**
Primary difference: **contiguous run** (LSM-Tree) versus **fragmented leaves** (B-Tree)

Straightforward approach



copy data from B-Tree leaves to create contiguous bottom run of an LSM tree; **re-construct indexes** (bloom filters)

Optimized Approach



LSM tree **"page IDs" mapped to B-Tree's physical pages**

Cost Model to Choose the Optimal LSM-Tree to B-Tree Transition

We compare the IO costs of the two transition approaches described above.

$$\text{Sort-Merge Cost} = \sum_{i=1}^{\text{num levels}} \left\lceil \frac{\text{bytes in } i\text{th level}}{\text{page size}} \right\rceil \cdot \left(1 + \frac{\text{write cost}}{\text{read cost}}\right)$$

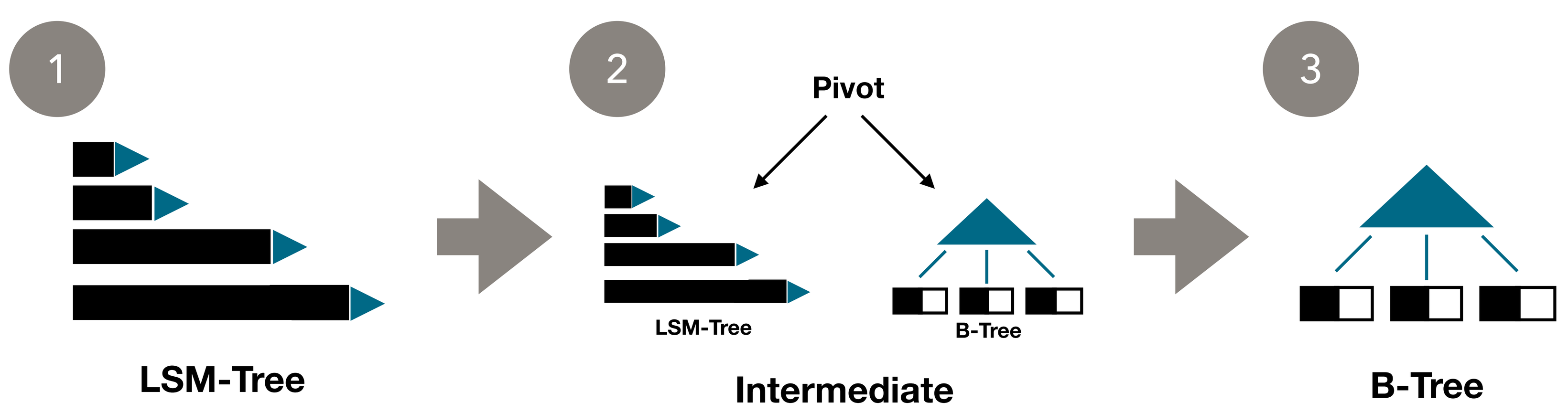
$$\text{Batch-Insert Cost} = \left\lceil \frac{\text{bytes in lowest level}}{\text{page size}} \right\rceil + \sum_{i=1}^{\text{num levels}-1} \text{bytes in } i\text{th level} \cdot \left(1 + 2 \cdot \frac{\text{write cost}}{\text{read cost}}\right)$$

Denoting ϕ as the ratio of IO write to read cost, d as the entry size in bytes, and p as the number of entries per page, we find an elegant condition for when we ought to prefer the batch-insert algorithm over the sort-merge algorithm.

$$\frac{\text{bytes in upper levels}}{\text{bytes in lowest level}} < \frac{d \cdot \phi}{p + (2 \cdot p - d) \cdot \phi}$$

Gradual Transitions Enable Low Overhead

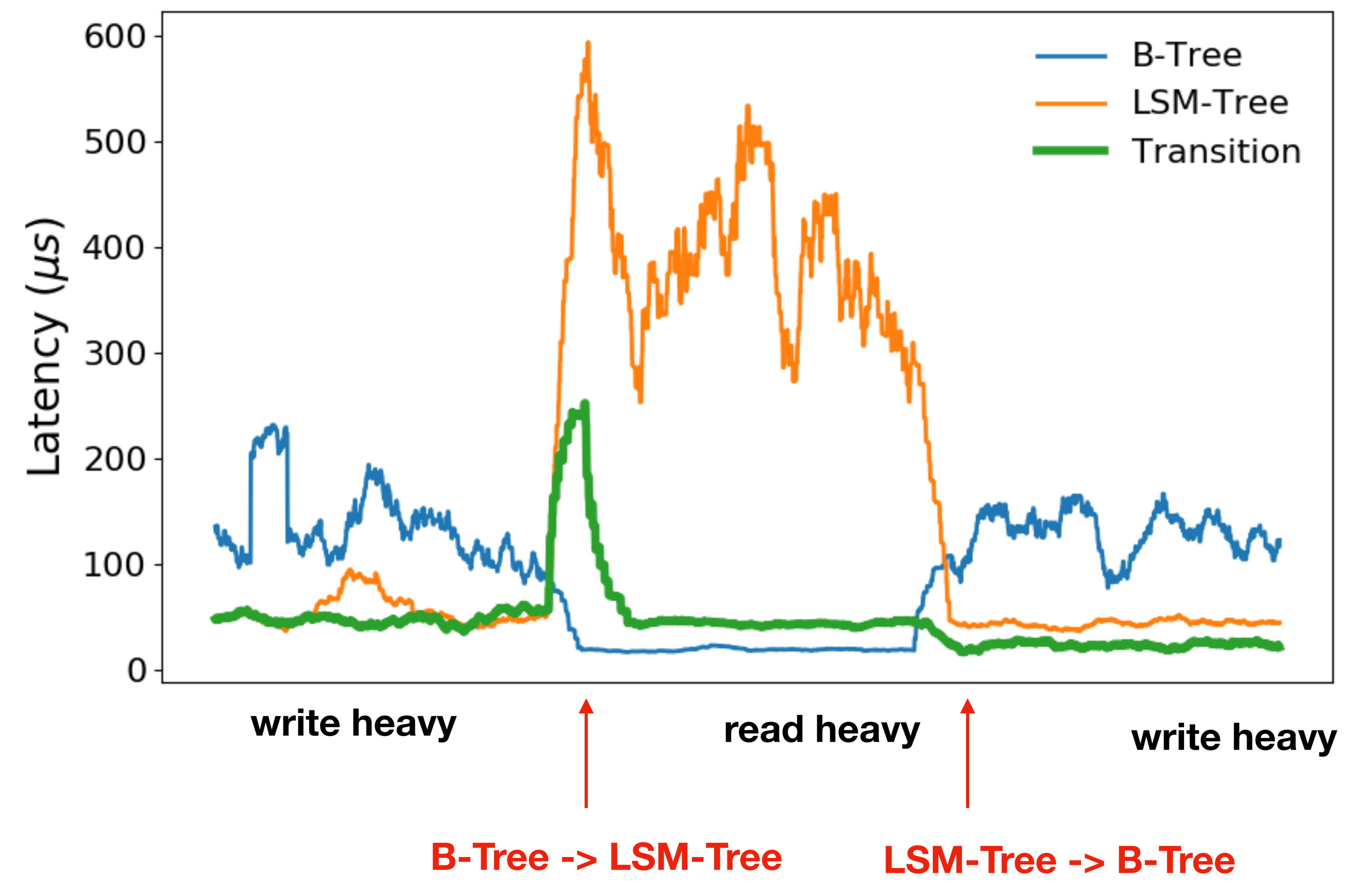
When transitioning from an LSM-Tree to a B-Tree, the transition cost can be **amortized over an arbitrary number of steps**. We maintain a **hybrid key-value store** to handle queries while the transition is in progress.



When transitioning from a B-Tree to an LSM-Tree, gradual transitions aren't necessary since this is a **cheap, in-memory operation**.

Transitioning Outperforms B-Trees and LSM Trees

Our implementation of this hybrid data structure proves that transitioning databases can provide superior query performance on dynamic workloads than classic LSM or B-Trees can.



HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

daslab.seas.harvard.edu

crimsondb.org

DASlab