

L-Store: A Real-time OLTP and OLAP System

Mohammad Sadoghi[†]

Souvik Bhattacharjee[‡], Bishwaranjan Bhattacharjee[#], Mustafa Canim[#]

[†]Exploratory Systems Lab

[†] University of California, Davis

[‡] University of Maryland, College Park

[#] IBM T.J. Watson

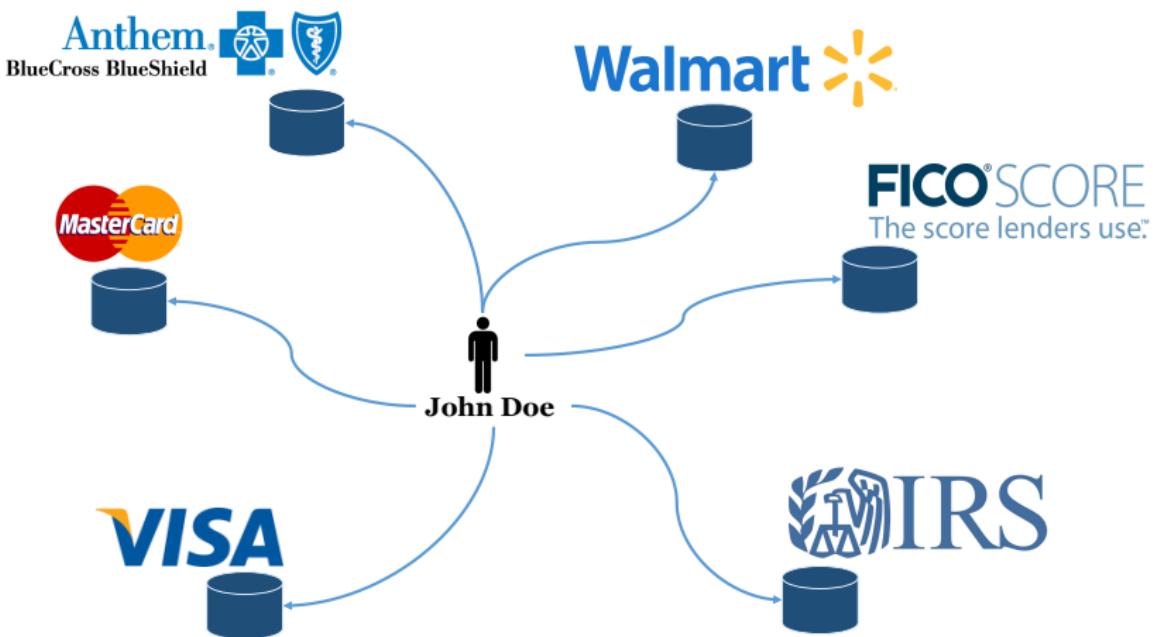
EDBT'18 (March 27, 2018)



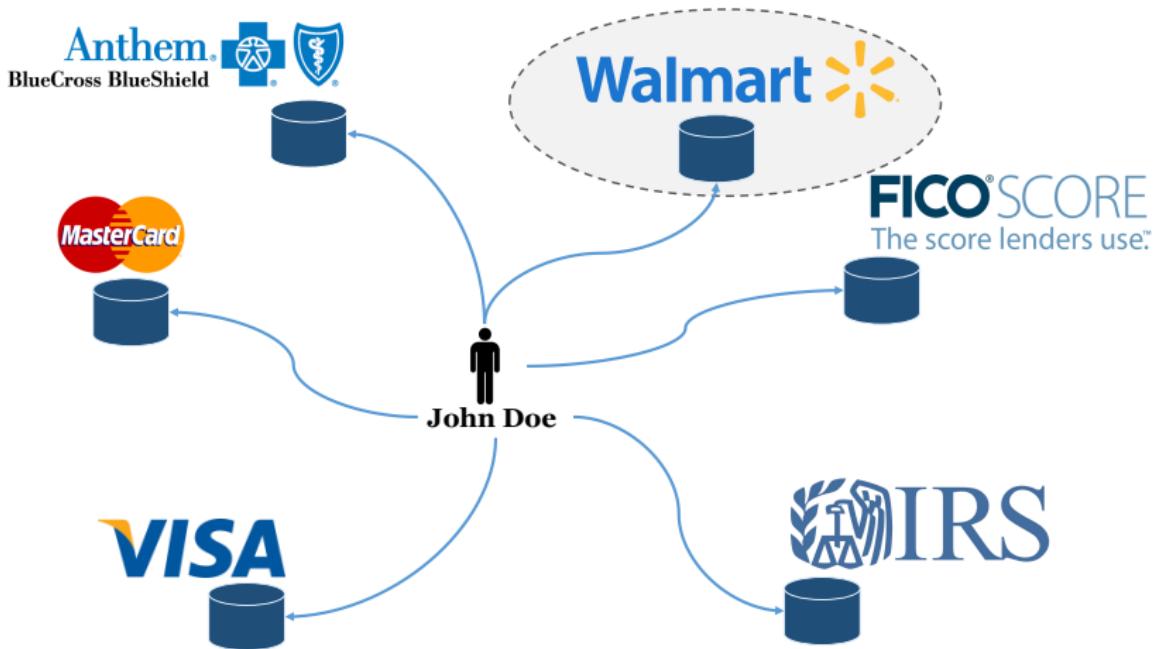
IBM Research



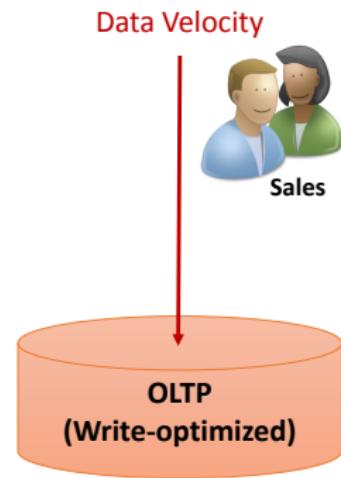
Data Management at Macroscale: The Four V's of Big Data



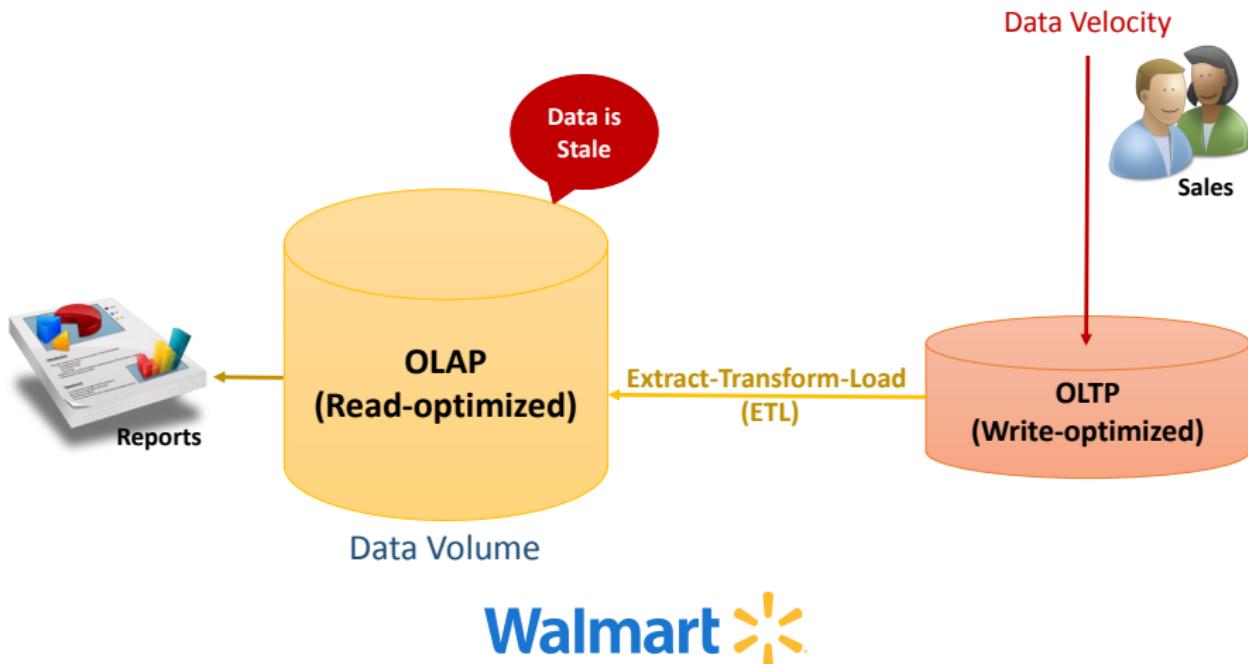
Data Management at Macroscale: The Four V's of Big Data



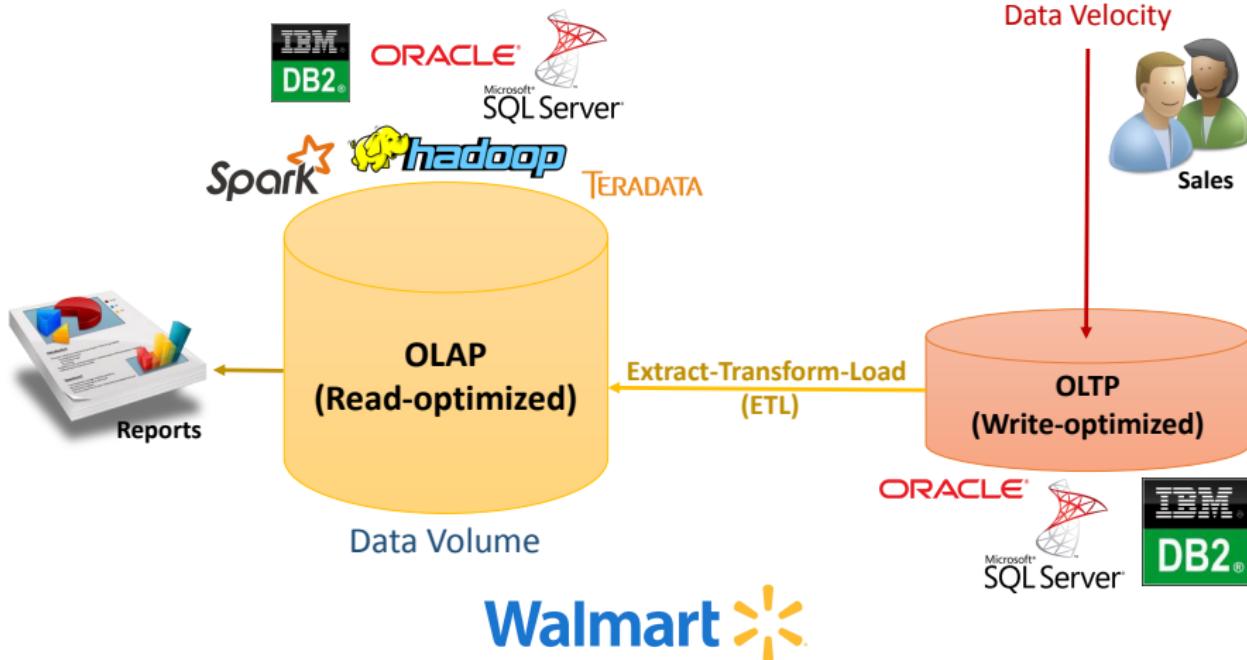
Data Management at Microscale: Volume & Velocity



Data Management at Microscale: Volume & Velocity

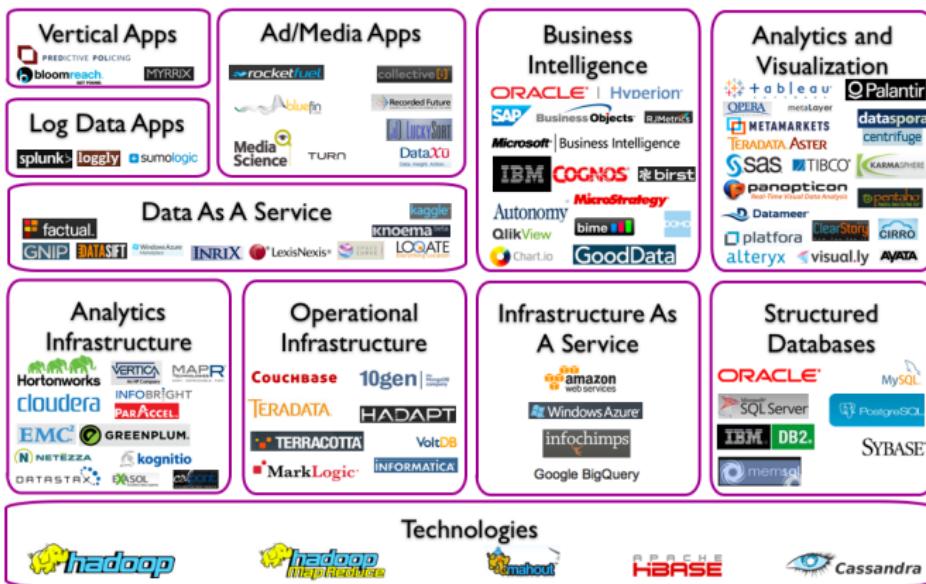


Data Management at Microscale: Volume & Velocity



One Size Does not Fit All As of 2012

Big Data Landscape

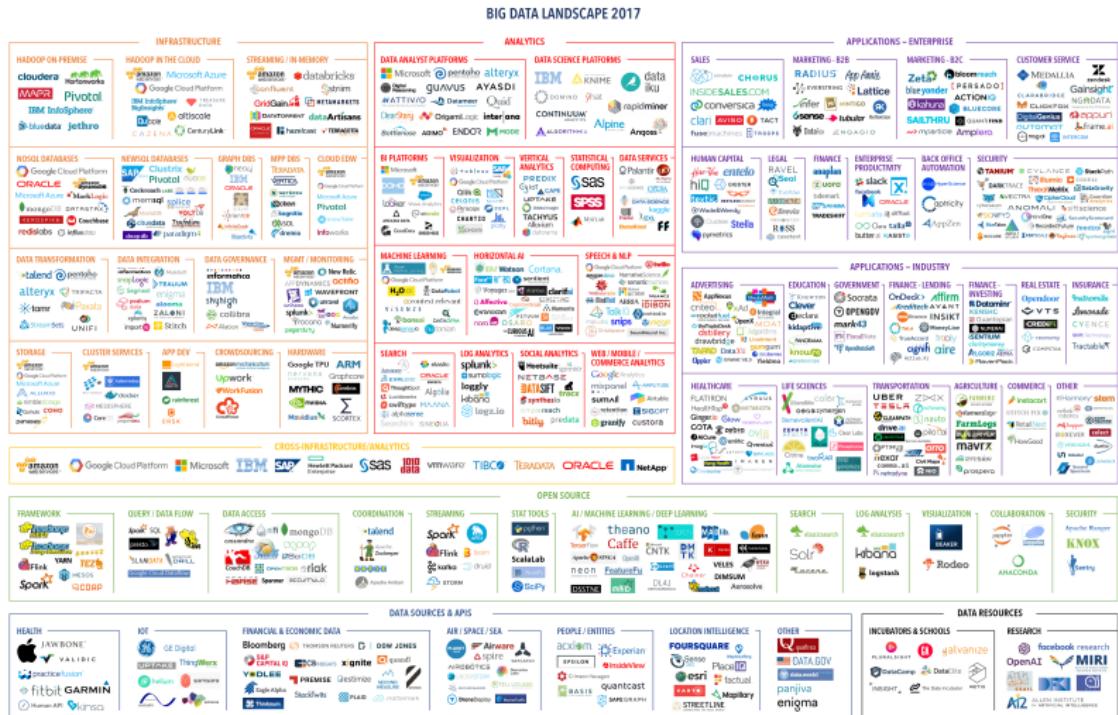


Copyright © 2012 Dave Feinleib

dave@vcddave.com

blogs.forbes.com/davefeinleib

One Size Does not Fit All As of 2017



142 Last updated 15/10/2017

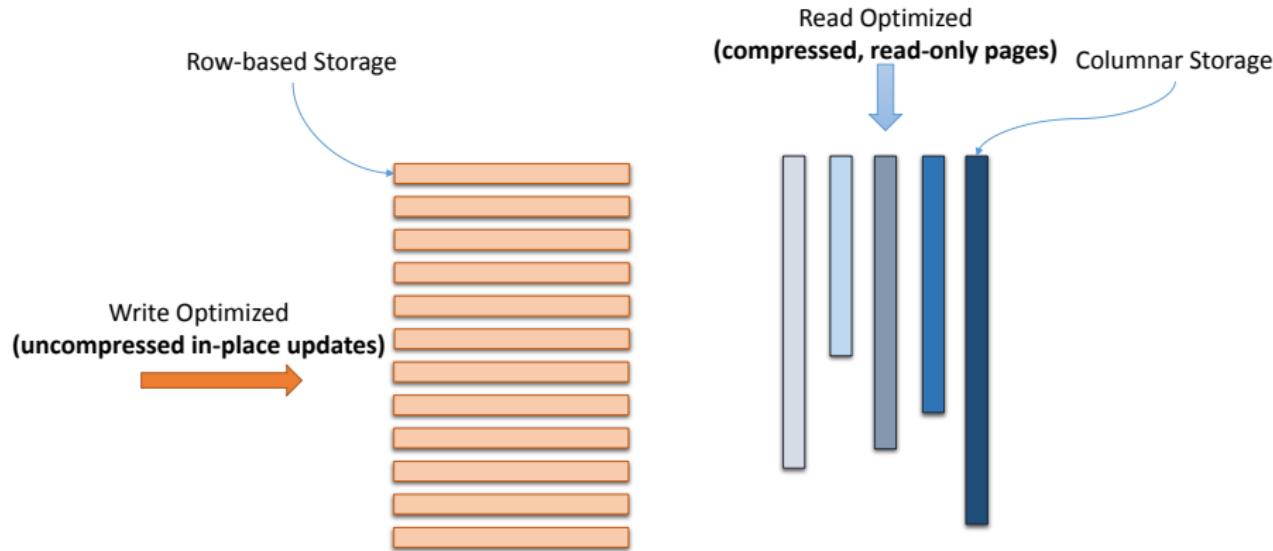
© Matt Turek (@mathtturk), Em Hao (@haoem), & FirstMark (@firstmarkcap).

www.ncbi.nlm.nih.gov/pmc/articles/PMC3737237/

Data Management at Microscale: Volume & Velocity



Storage Layout Conflict



Write-optimized (i.e., uncompressed & row-based) vs. read-optimized (i.e., compressed & column-based) layouts

Unifying OLTP and OLAP: Velocity & Volume Dimensions

Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).

Unifying OLTP and OLAP: Velocity & Volume Dimensions

Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).

Introducing a *lineage-based storage architecture*, a contention-free update mechanism over a native columnar storage in order to

Unifying OLTP and OLAP: Velocity & Volume Dimensions

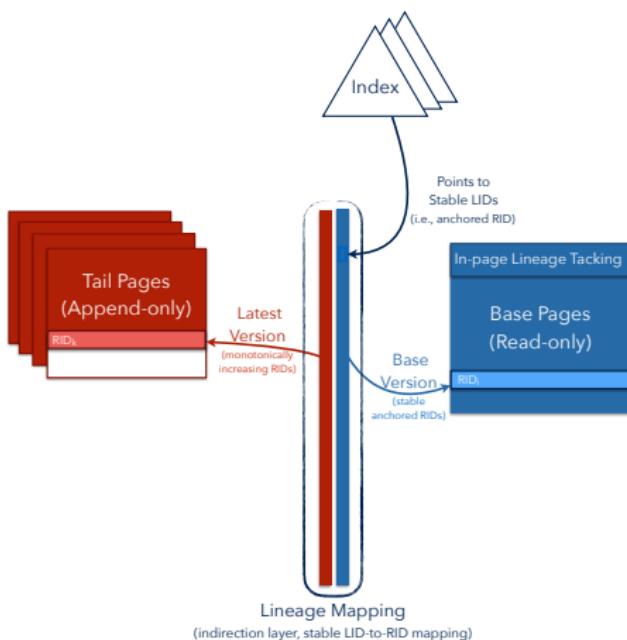
Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).

Introducing a *lineage-based storage architecture*, a contention-free update mechanism over a native columnar storage in order to

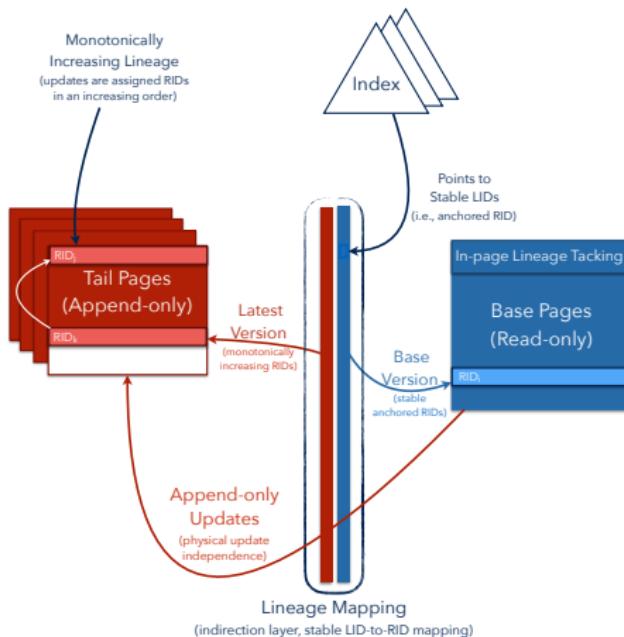
lazily and independently stage stable data from a write-optimized layout (i.e., OLTP) into a read-optimized layout (i.e., OLAP)

Lineage-based Storage Architecture (LSA): Intuition



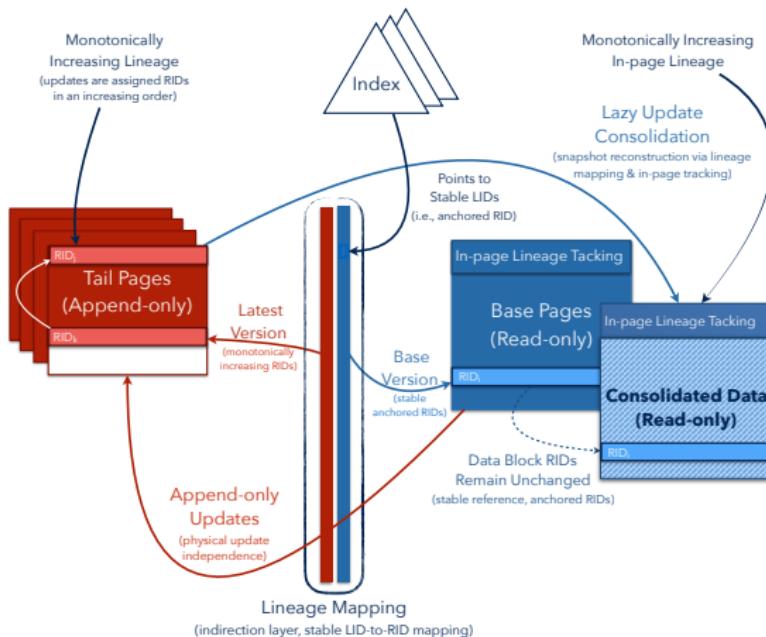
Physical Update Independence: De-coupling data & its updates
(reconstruction via in-page lineage tracking and lineage mapping)

Lineage-based Storage Architecture (LSA): Intuition



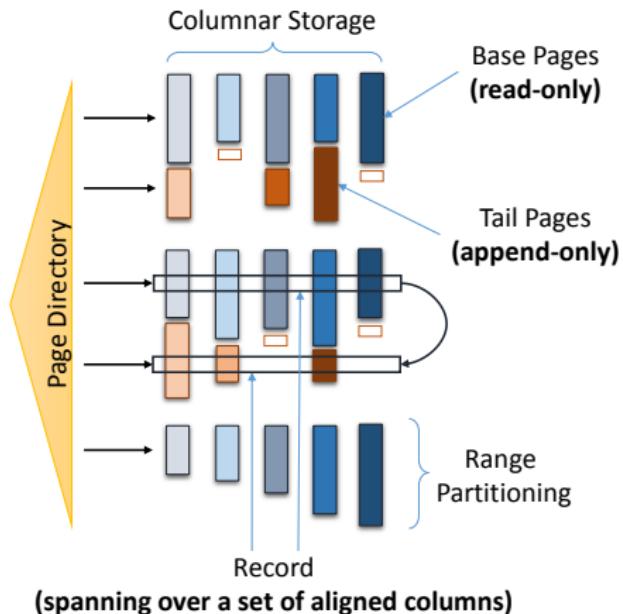
Physical Update Independence: De-coupling data & its updates
(reconstruction via in-page lineage tracking and lineage mapping)

Lineage-based Storage Architecture (LSA): Intuition



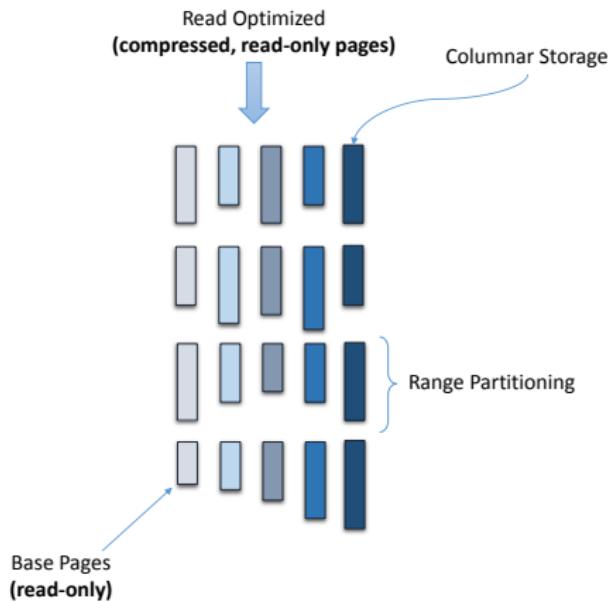
Physical Update Independence: De-coupling data & its updates
(reconstruction via in-page lineage tracking and lineage mapping)

Lineage-based Storage Architecture (LSA): Overview



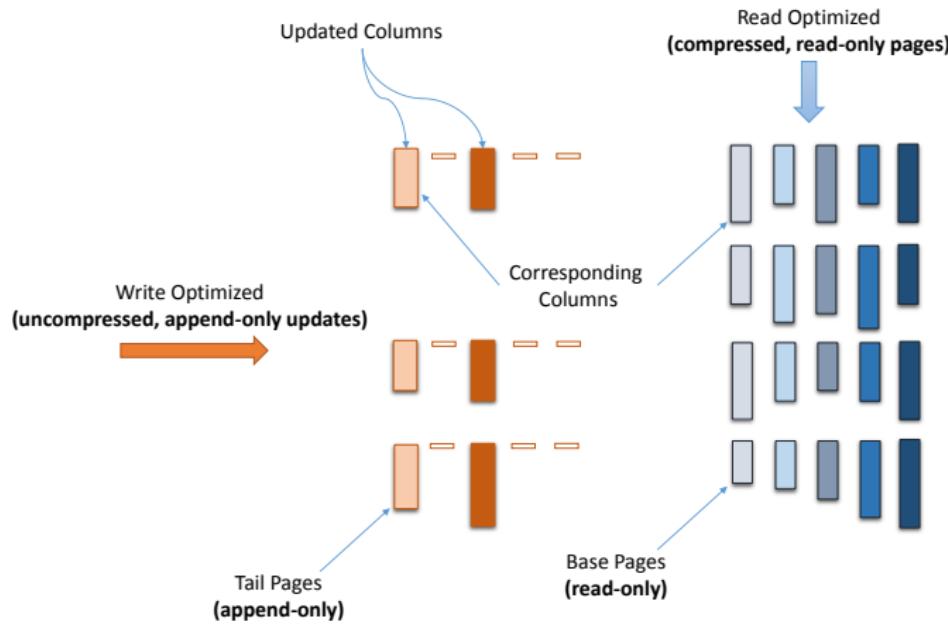
Overview of the lineage-based storage architecture
(**base pages** and **tail pages** are handled identically at the storage layer)

L-Store: Detailed Design



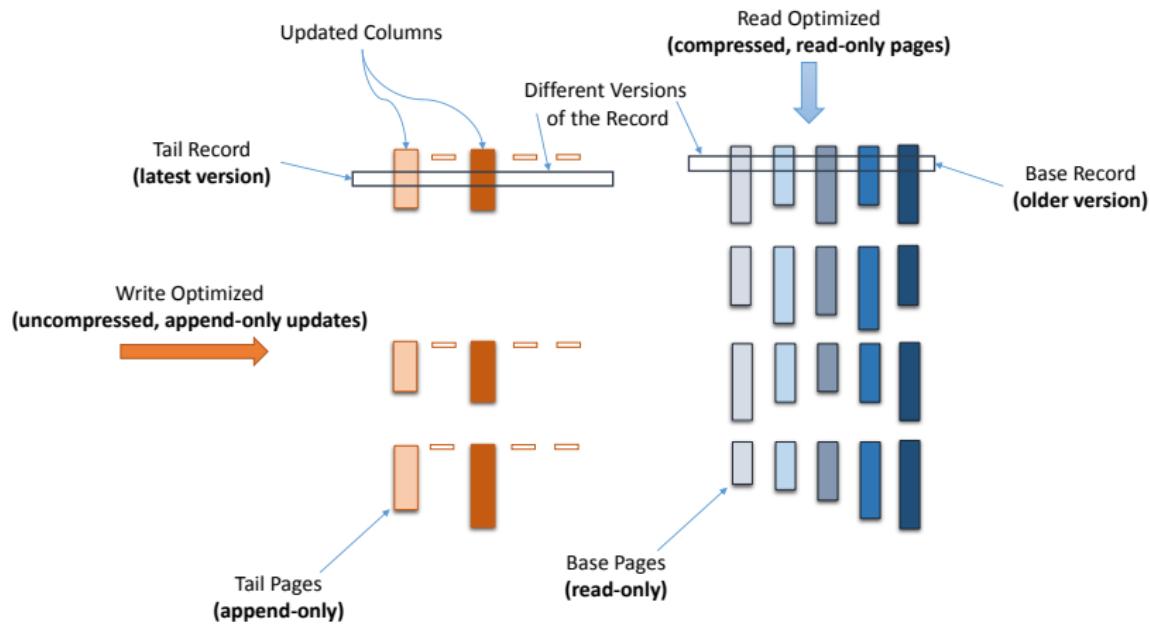
Records are range-partitioned and compressed into a set of ready-only **base pages** (accelerating analytical queries)

L-Store: Detailed Design



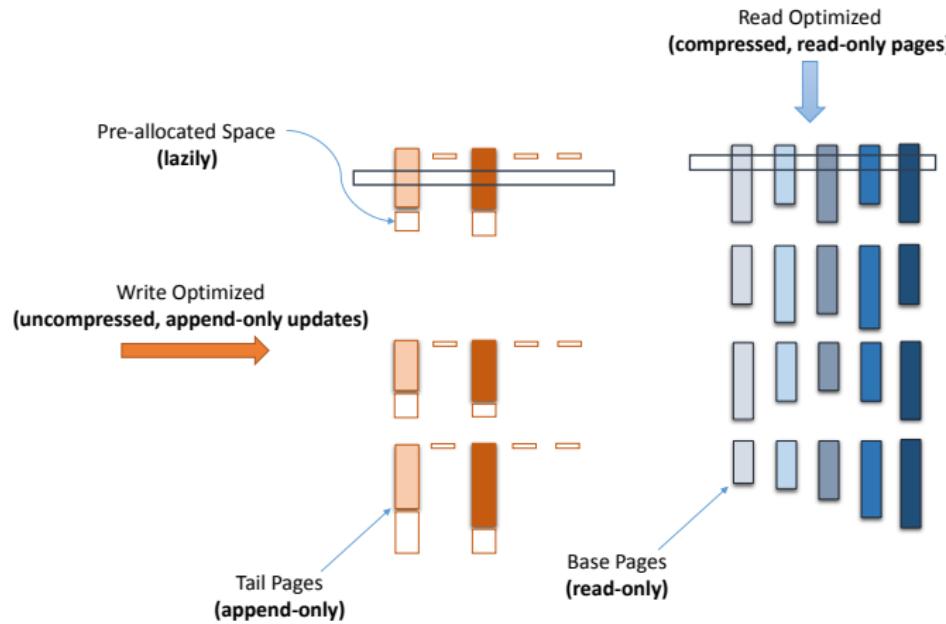
Recent updates for a range of records are clustered in their **tails pages** (transforming costly point updates into an amortized analytical-like query)

L-Store: Detailed Design



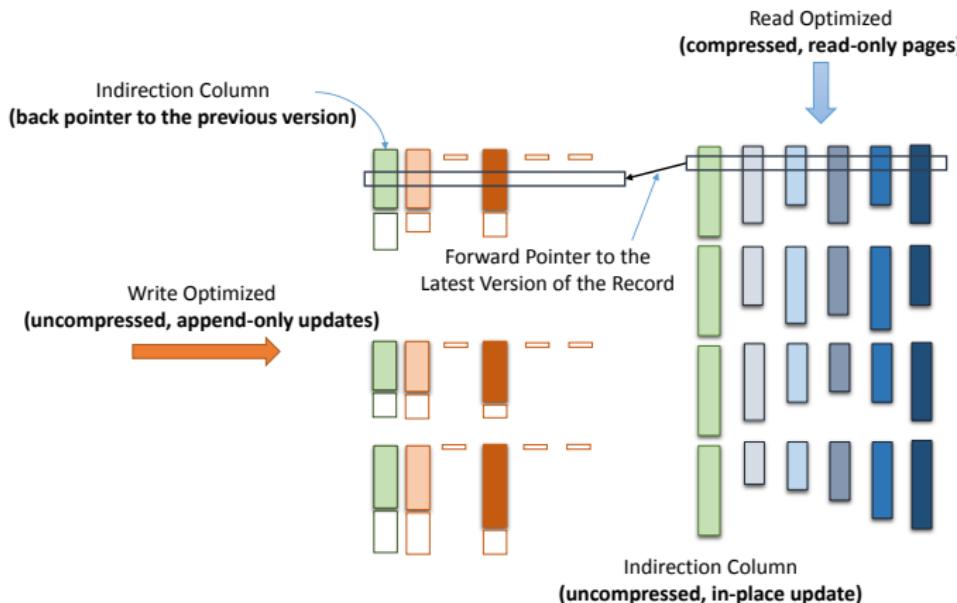
Recent updates for a range of records are clustered in their **tails pages** (transforming costly point updates into an amortized analytical-like query)

L-Store: Detailed Design



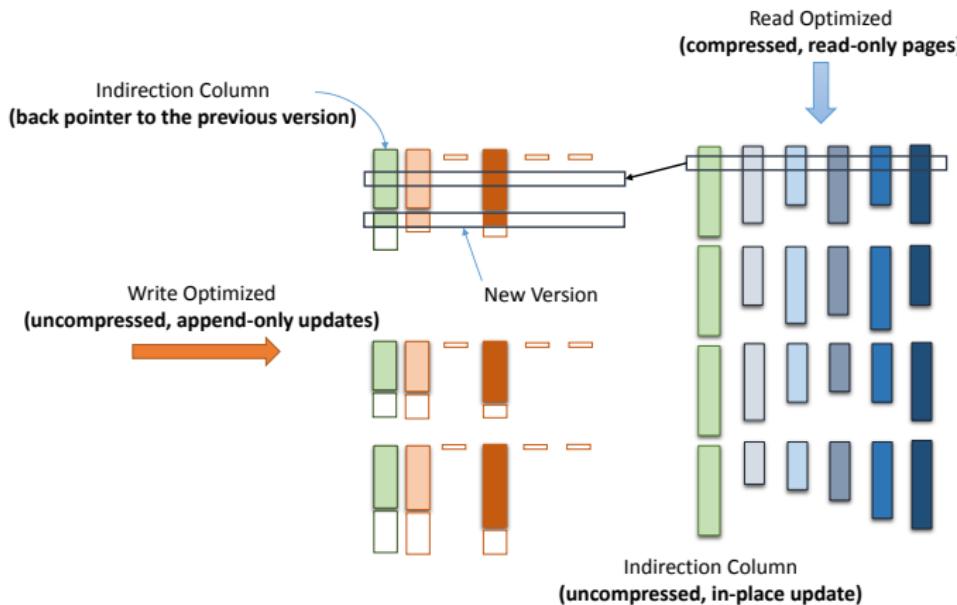
Recent updates are strictly appended, uncompressed in the pre-allocated space
(eliminating the read/write contention)

L-Store: Detailed Design



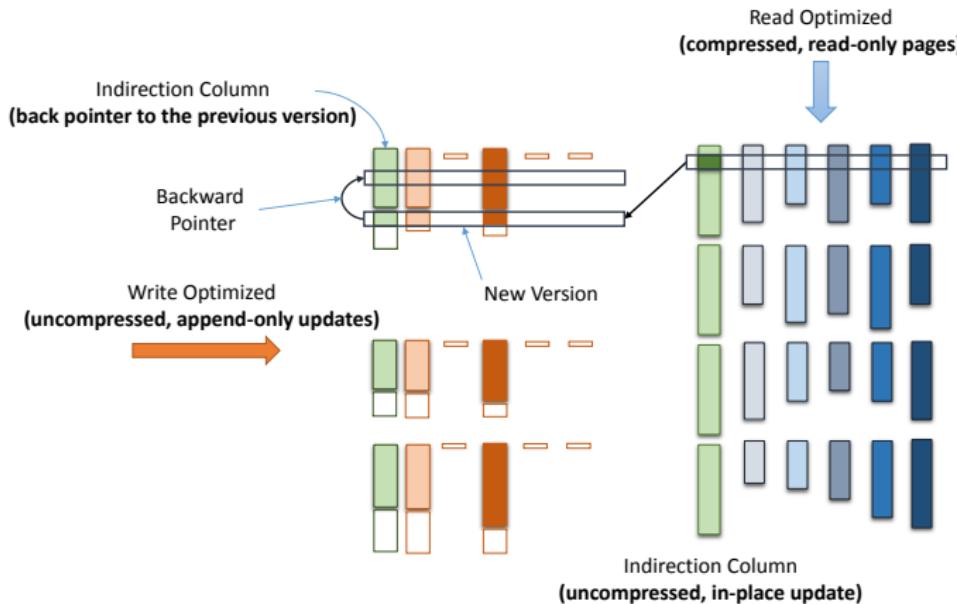
Achieving (at most) 2-hop access to the latest version of any record
(avoiding read performance deterioration for point queries)

L-Store: Detailed Design



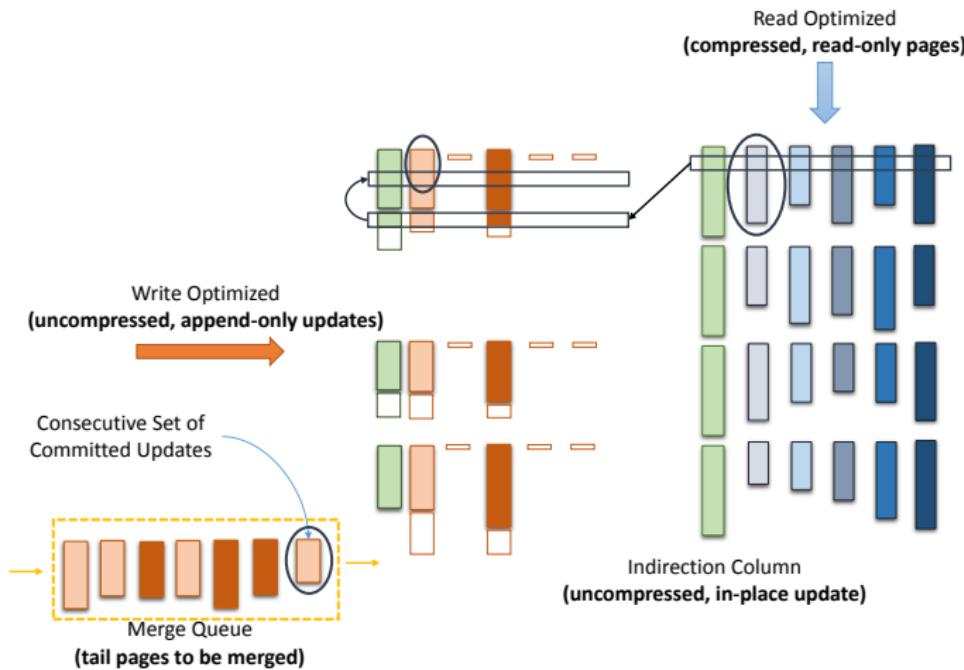
Achieving (at most) 2-hop access to the latest version of any record
(avoiding read performance deterioration for point queries)

L-Store: Detailed Design



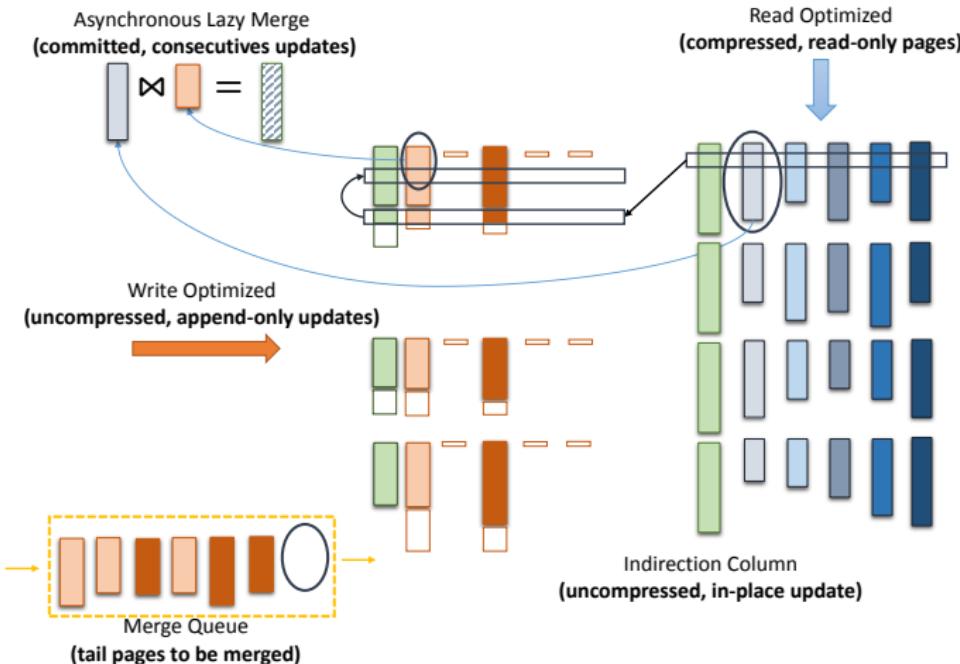
Achieving (at most) 2-hop access to the latest version of any record
(avoiding read performance deterioration for point queries)

L-Store: Contention-free Merge



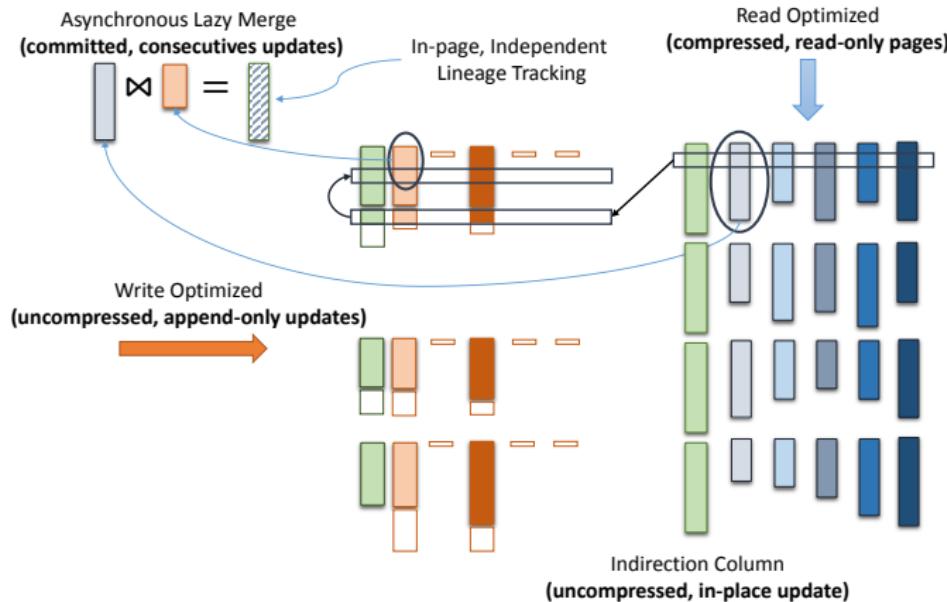
Contention-free merging of only stable data: read-only and committed data
(no need to block on-going and new transactions)

L-Store: Contention-free Merge



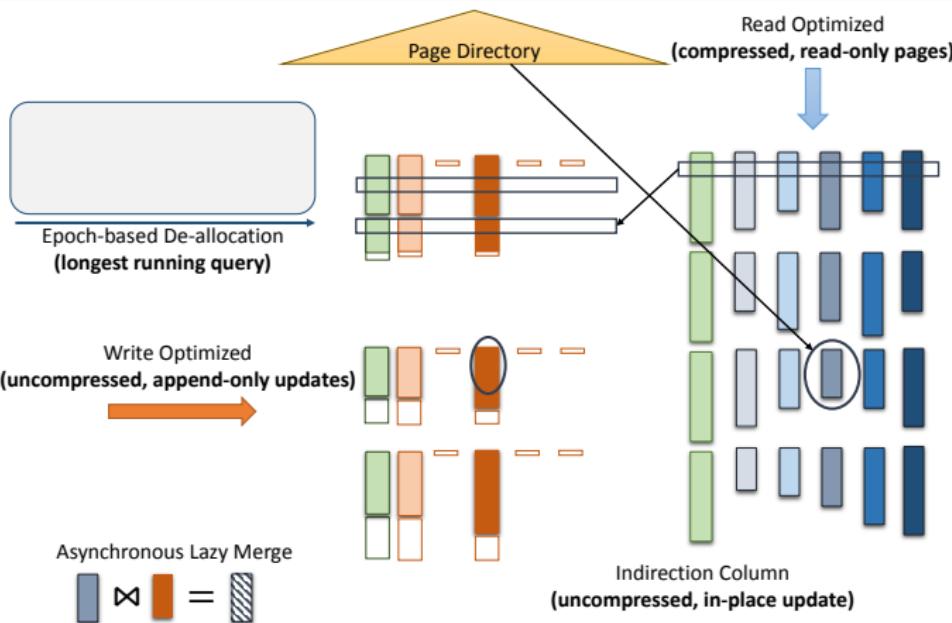
Lazy independent merging of **base pages** with their corresponding **tail pages**
(resembling a local left outer-join of the base and tail pages)

L-Store: Contention-free Merge



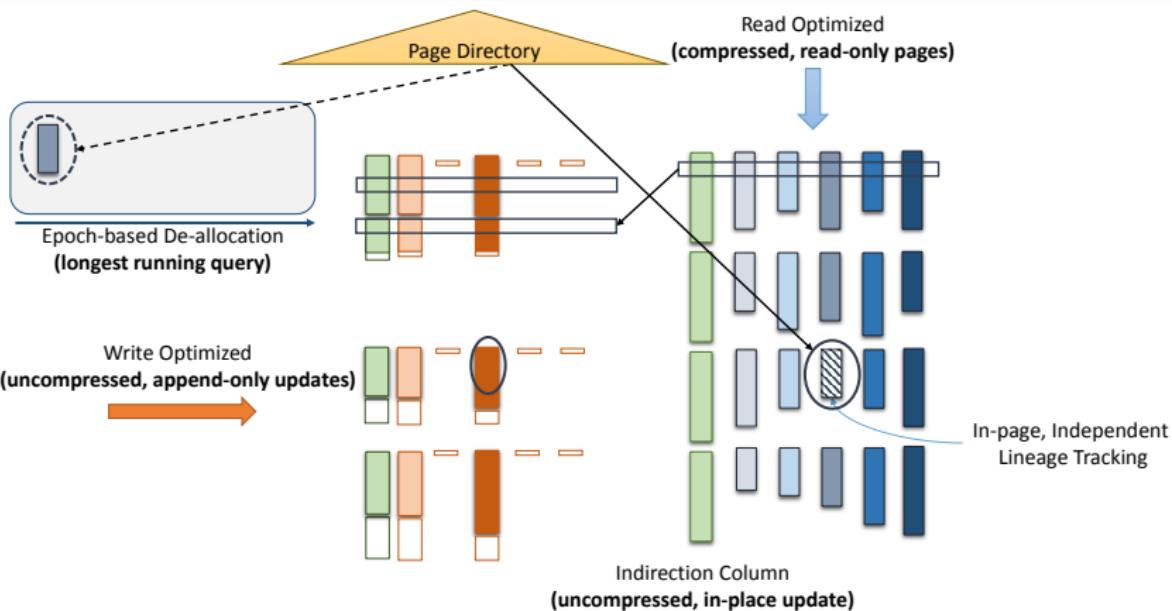
Independently tracking the lineage information within every page
(no need to coordinate merges among different columns of the same records)

L-Store: Epoch-based Contention-free De-allocation



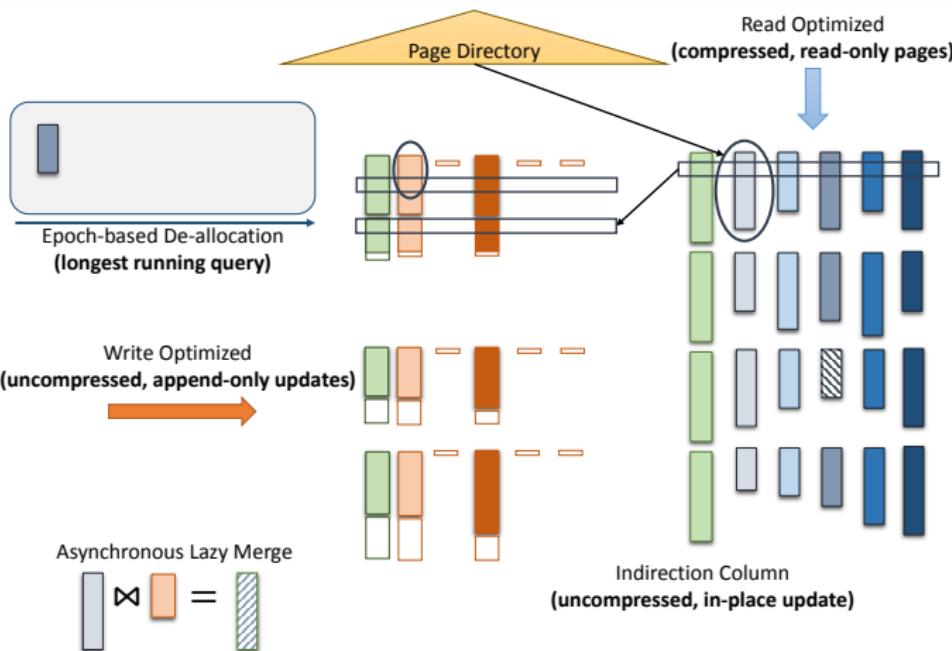
Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)

L-Store: Epoch-based Contention-free De-allocation



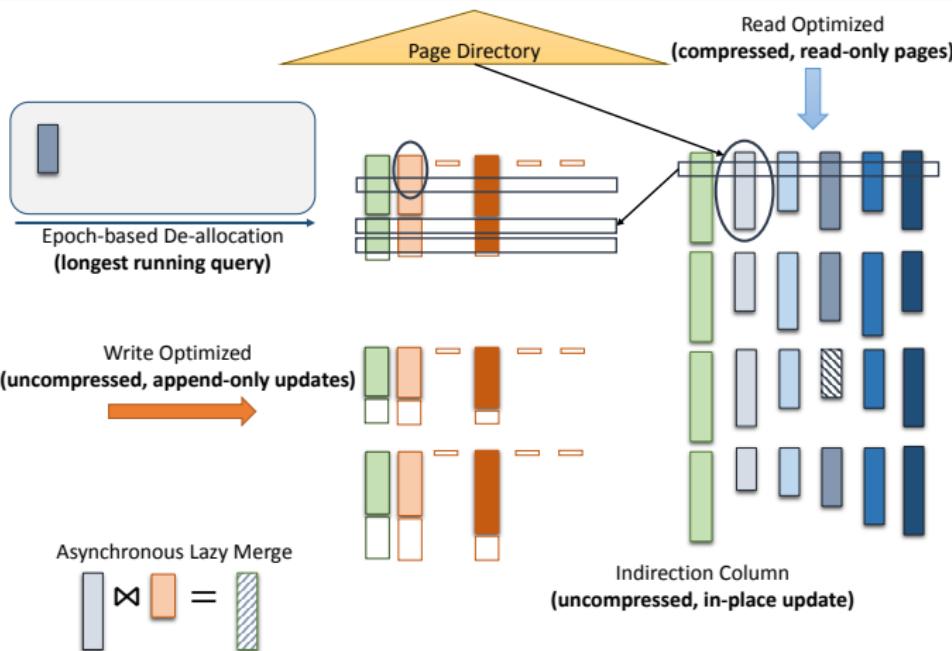
Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)

L-Store: Epoch-based Contention-free De-allocation



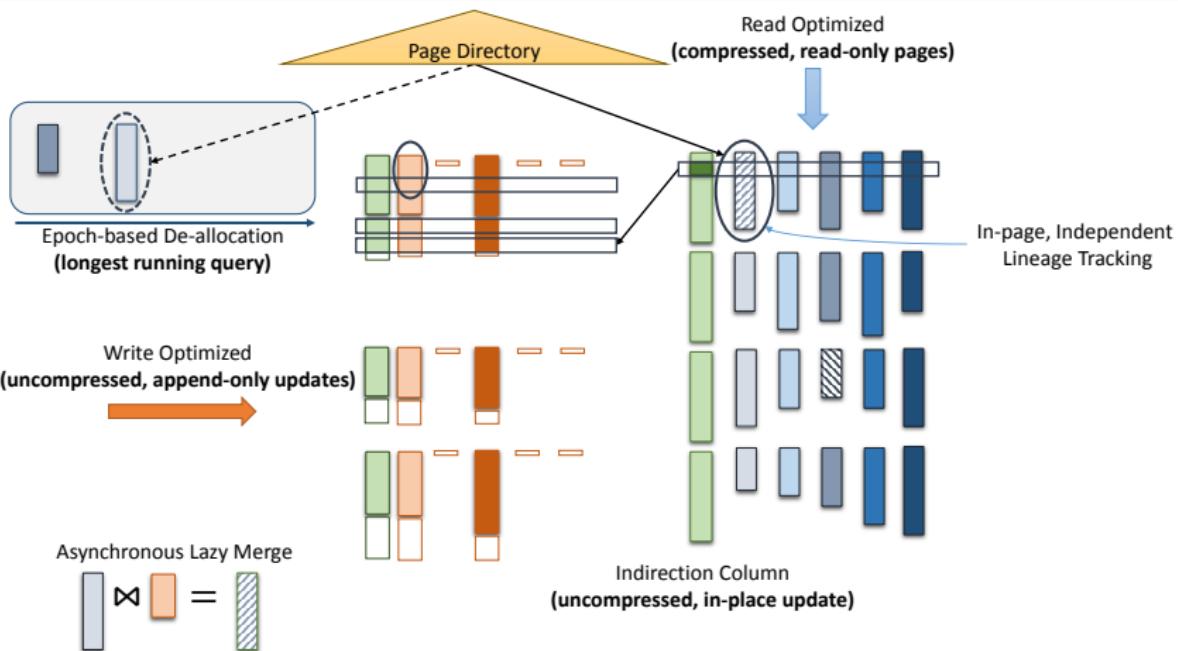
Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)

L-Store: Epoch-based Contention-free De-allocation



Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)

L-Store: Epoch-based Contention-free De-allocation



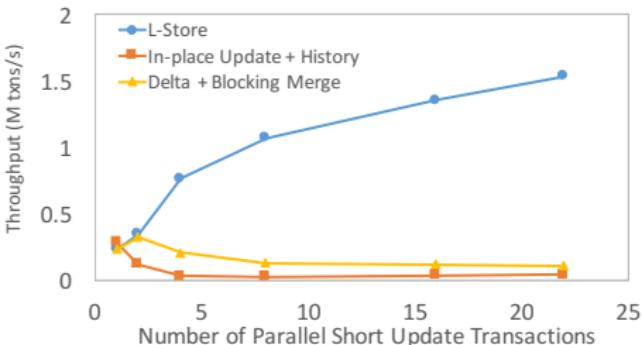
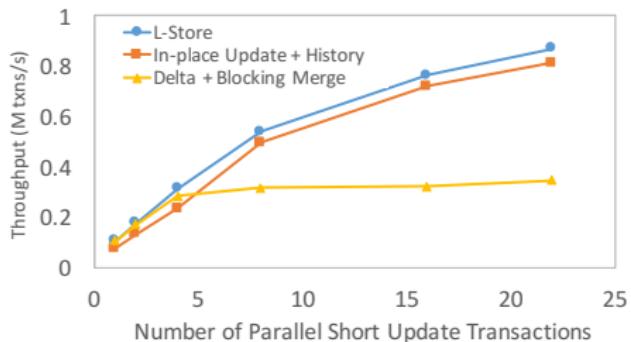
Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)

Experimental Analysis

Experimental Settings

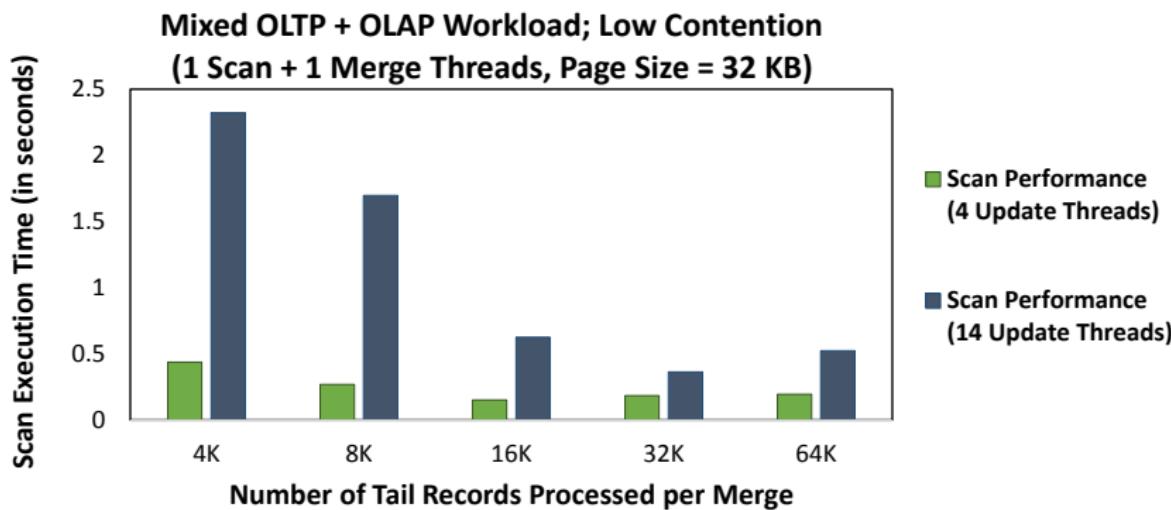
- Hardware:
 - 2 × 6-core Intel(R) Xeon(R) CPU E5-2430 @ 2.20GHz, 64GB, 15 MB L3 cache
- Workload: Extended Microsoft Hekaton Benchmark
 - Comparison with *In-place Update + History* and *Delta + Blocking Merge*
 - Effect of varying contention levels
 - Effect of varying the read/write ratio of short update transactions
 - Effect of merge frequency on scan
 - Effect of varying the number of short update vs. long read-only transactions
 - Effect of varying L-Store data layouts (row vs. columnar)
 - Effect of varying the percentage of columns read in point queries
 - Comparison with log-structured storage architecture (*LevelDB*)

Effect of Varying Contention Levels



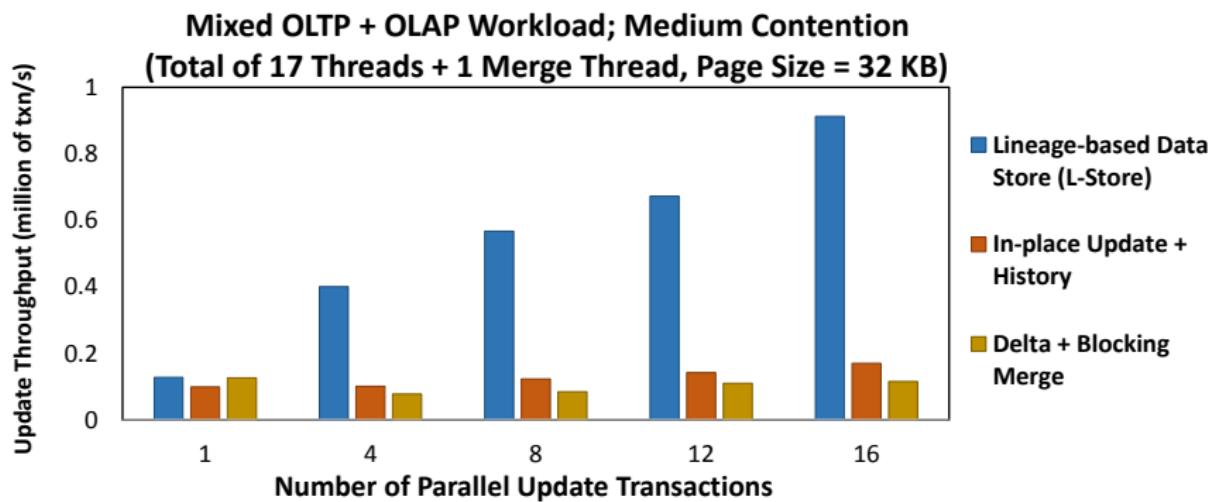
Achieving up to **40×** as increasing the update contention

Effect of Merge Frequency on Scan Performance



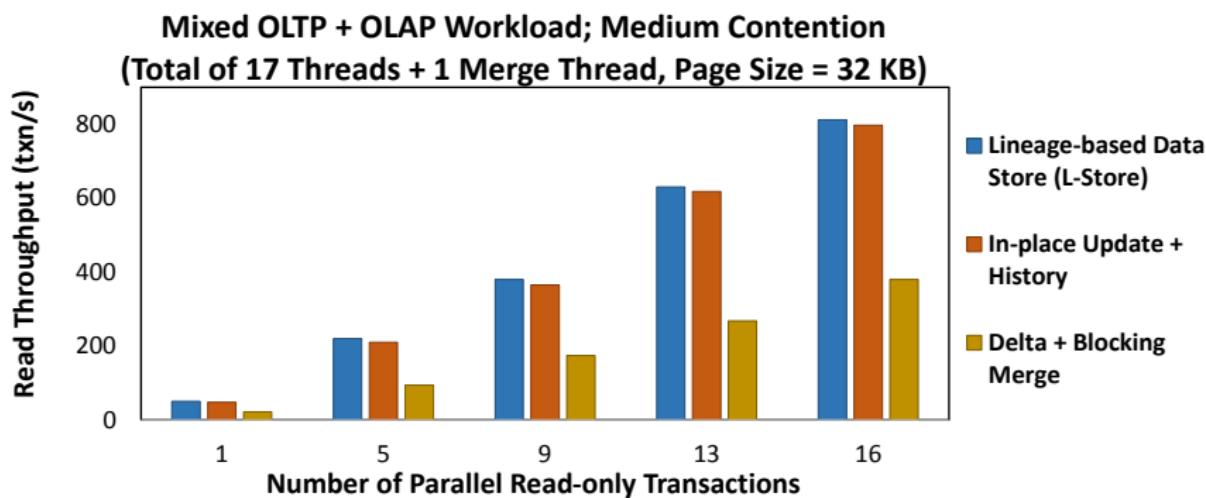
Merge process is essential in maintaining efficient scan performance

Effect of Mixed Workloads: Update Performance



Eliminating latching & locking results in a substantial performance improvement

Effect of Mixed Workloads: Read Performance



Coping with tens of update threads with a single merge thread

L-Store Key Contributions

- Unifying OLAP & OLTP by introducing lineage-based storage architecture (LSA)
- LSA is a native multi-version, columnar storage model that lazily & independently stages data from a write-optimized layout into a read-optimized one
- Contention-free merging of only stable data without blocking ongoing or incoming transactions
- Contention-free page de-allocation without draining ongoing transactions
- L-Store outperforms in-place update & delta approaches by factor of up to **8×** on mixed OLTP/OLAP workloads and up to **40×** on update-intensive workloads

Questions?
Thank you!

Exploratory Systems Lab (ExpoLab)
Website: <https://msadoghi.github.io/>