

# L-Store: Lineage-based Storage Architectures

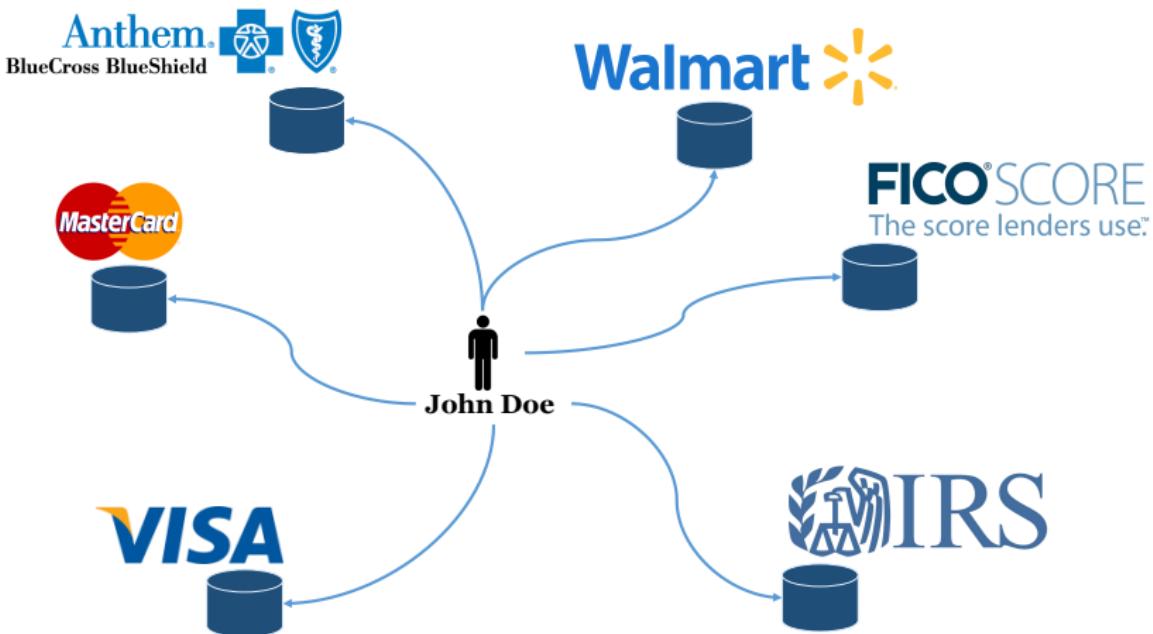
## ECS165A: Winter 2023

Slides are adopted from Sadoghi, et al.

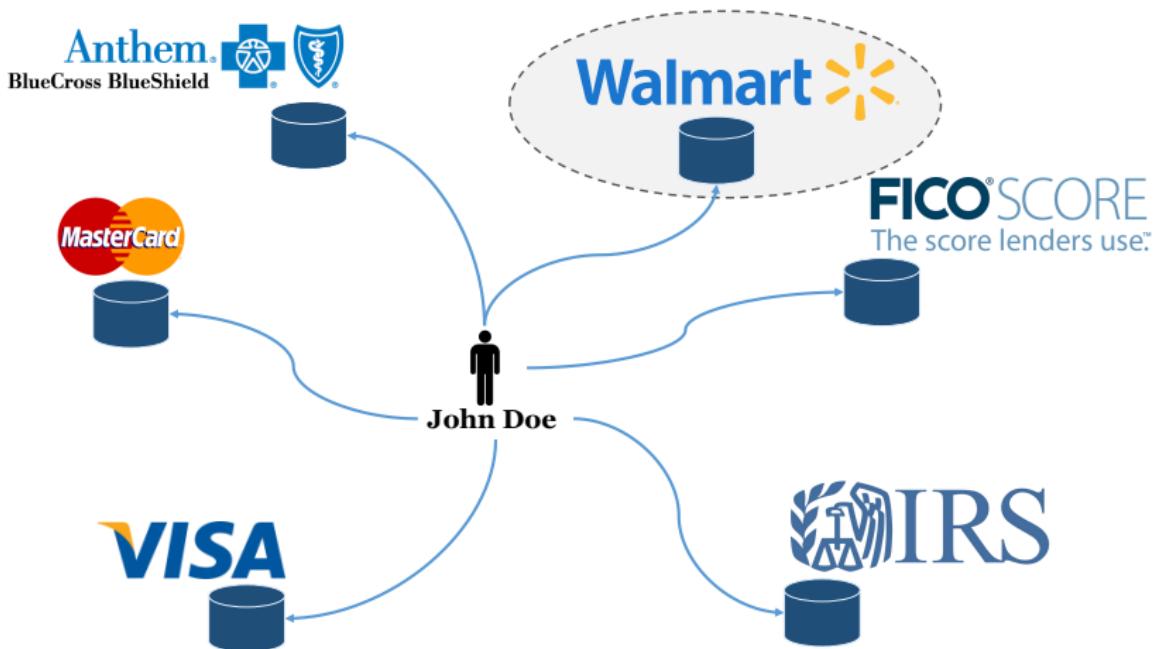
*L-Store: A Real-time OLTP and OLAP System, EDBT'18*



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



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

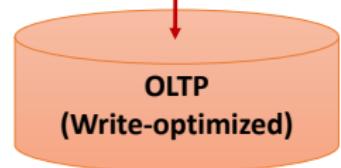


# Data Management at Microscale: Volume & Velocity

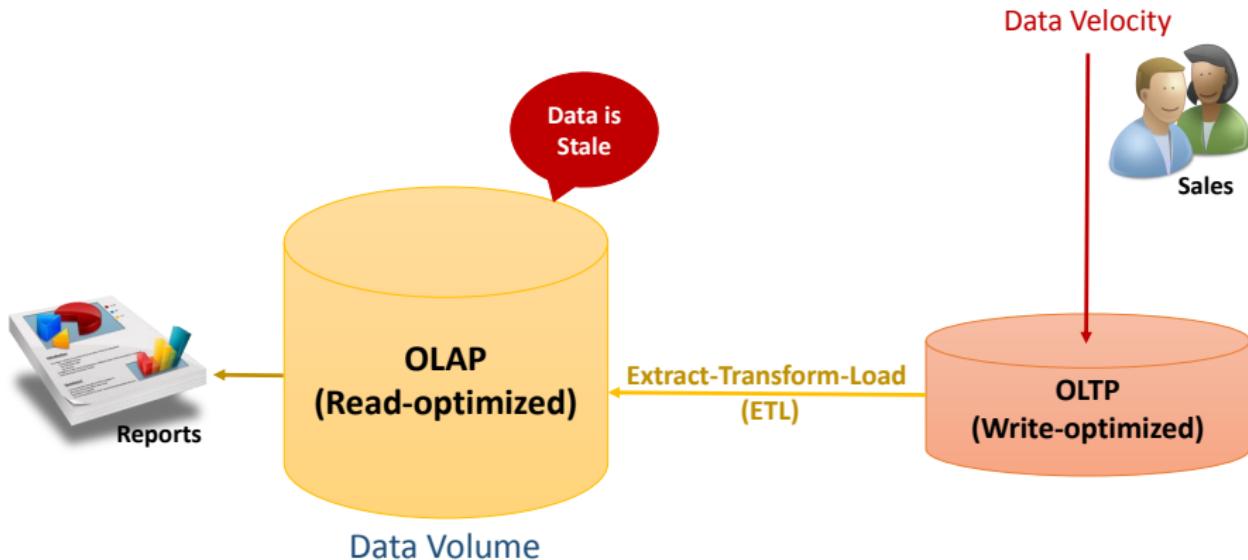
Data Velocity



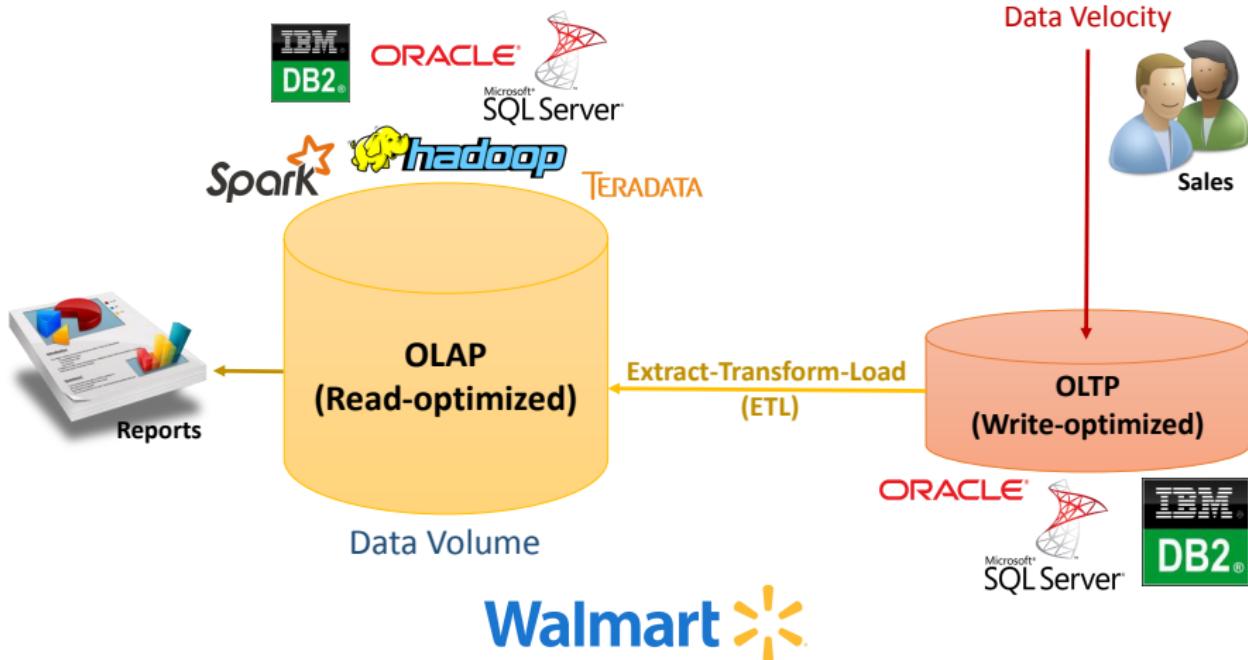
Sales



# Data Management at Microscale: Volume & Velocity

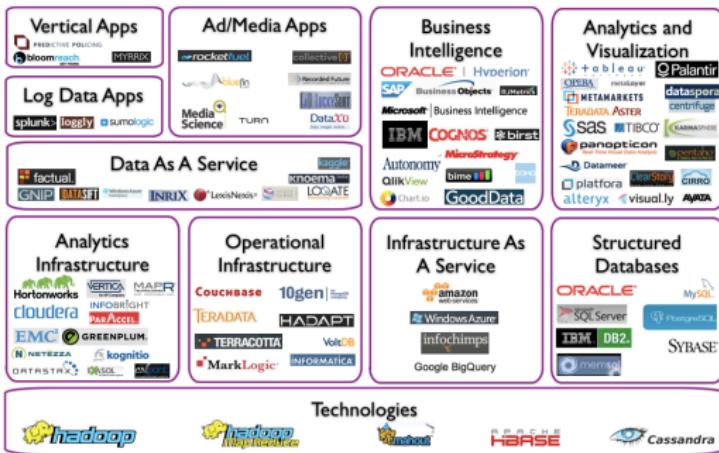


# Data Management at Microscale: Volume & Velocity



# One Size Does not Fit All As of 2012

## Big Data Landscape

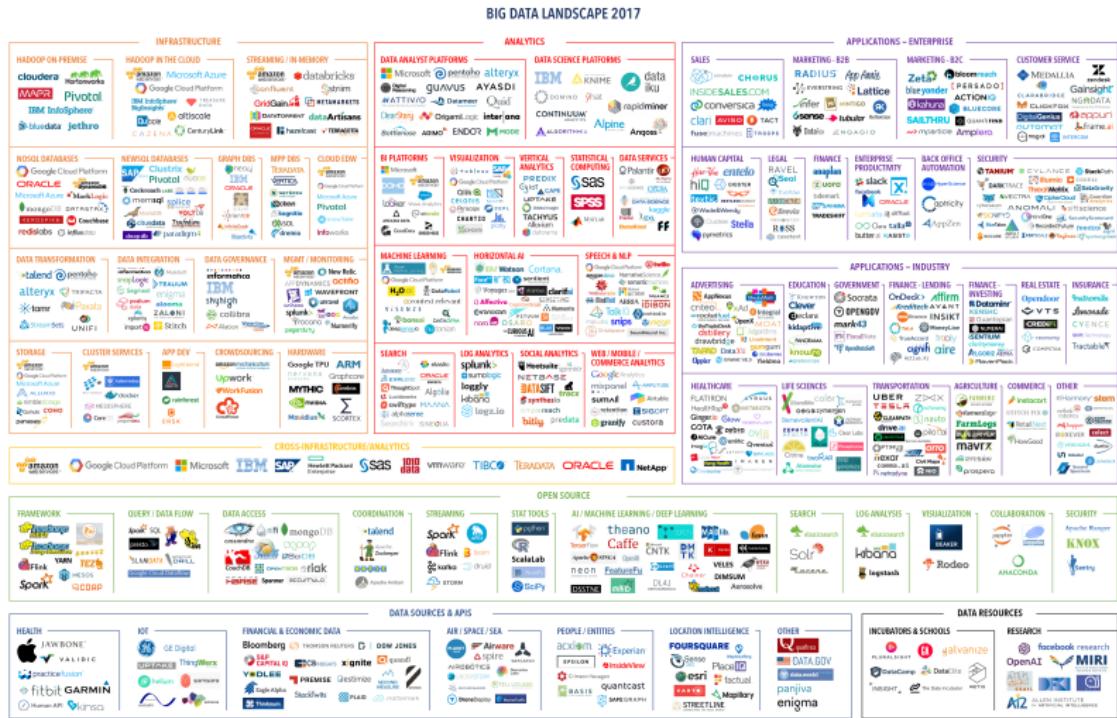


Copyright © 2012 Dave Feinleib

dave@vc.dave.com

blogs.forbes.com/davefeinleib

## One Size Does not Fit All As of 2017

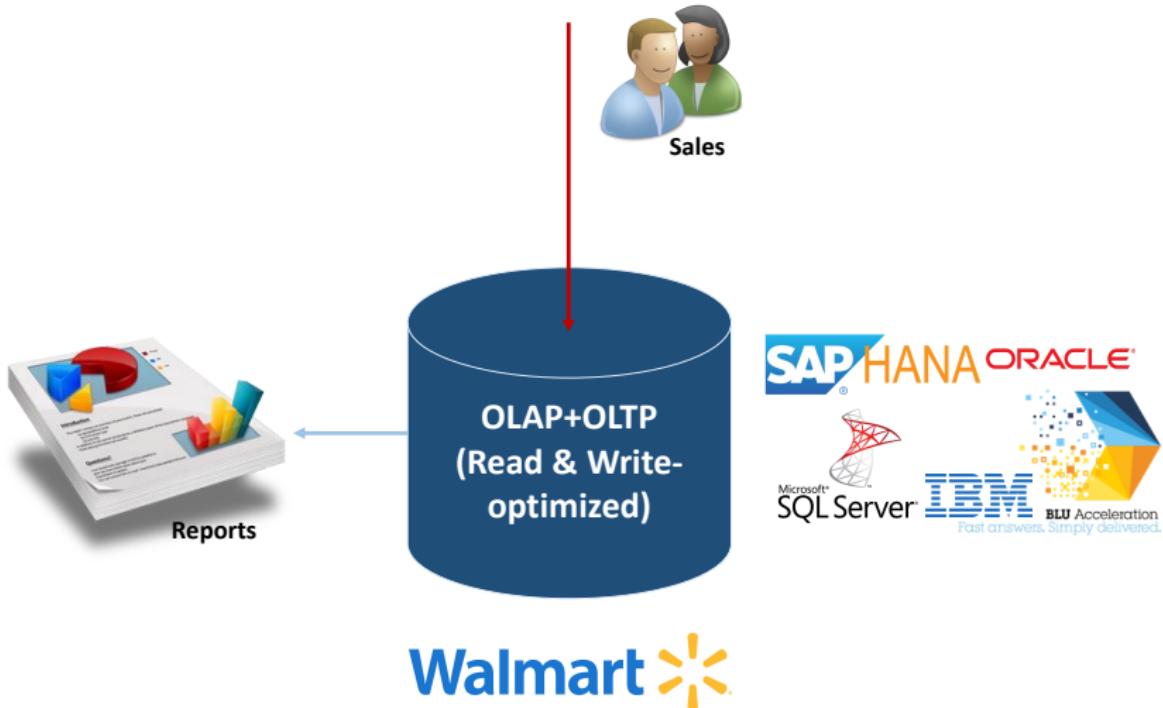


S2 - Last updated 5/3/2017

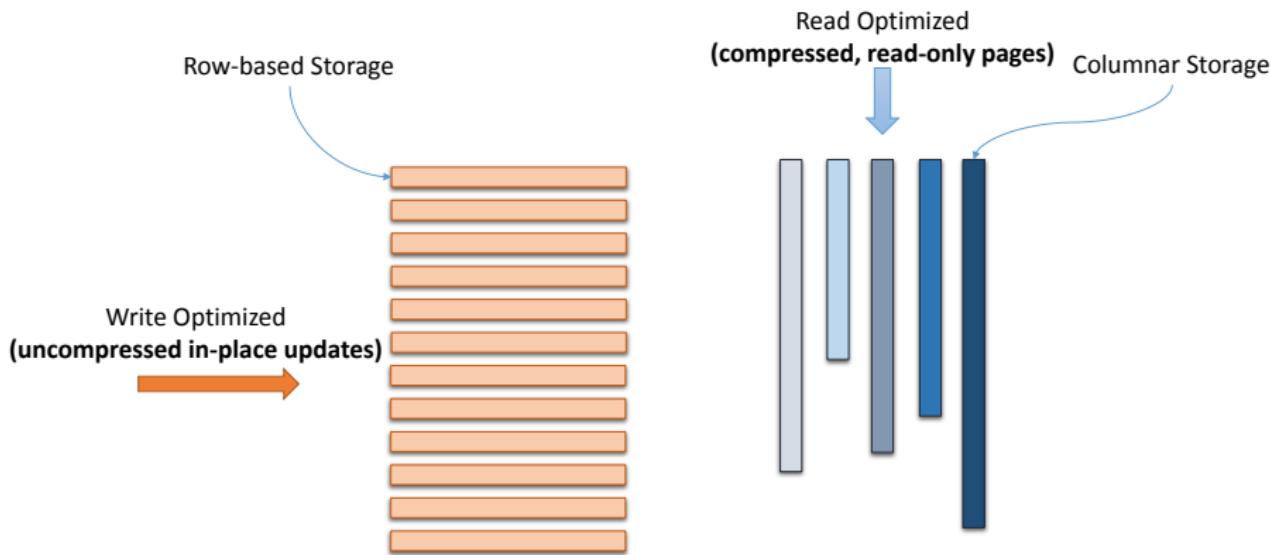
© Matt Turck (@mattturck), Jim Hao (@jimhao), & FirstMark (@firstmarkcap).

[mattturck.com/bio/data2017](http://mattturck.com/bio/data2017)

# 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

# Reducing Index maintenance: Velocity Dimension

## Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.

# Reducing Index maintenance: Velocity Dimension

## Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.

Extending storage hierarchy (using fast non-volatile memory) with *an extra level of indirection* in order to

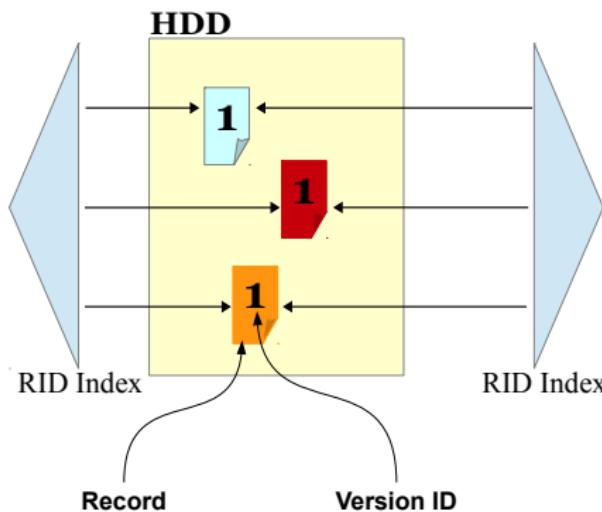
# Reducing Index maintenance: Velocity Dimension

## Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.

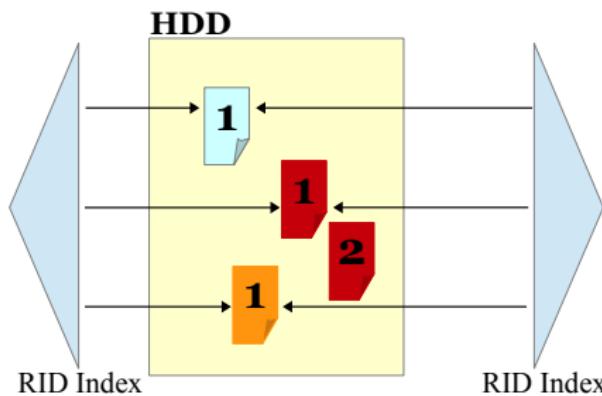
Extending storage hierarchy (using fast non-volatile memory) with *an extra level of indirection* in order to  
**Decouple Logical and Physical Locations of Records to Reduce Index Maintenance**

# Traditional Multi-version Indexing: Updating Records



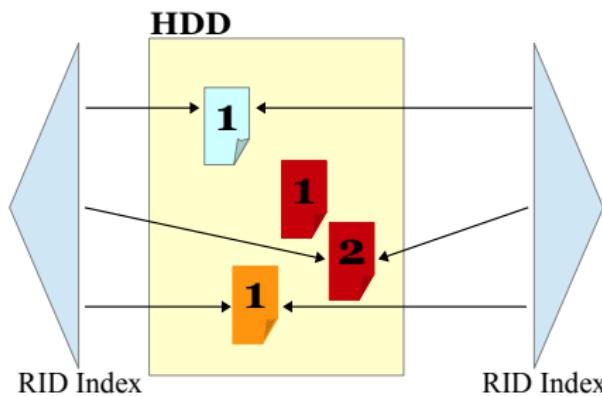
Updating random leaf pages

# Traditional Multi-version Indexing: Updating Records



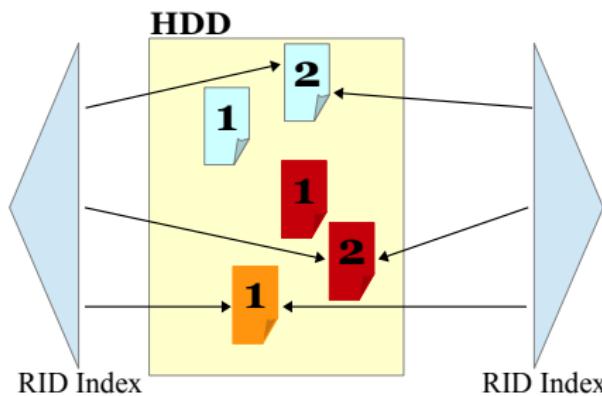
Updating random leaf pages

# Traditional Multi-version Indexing: Updating Records



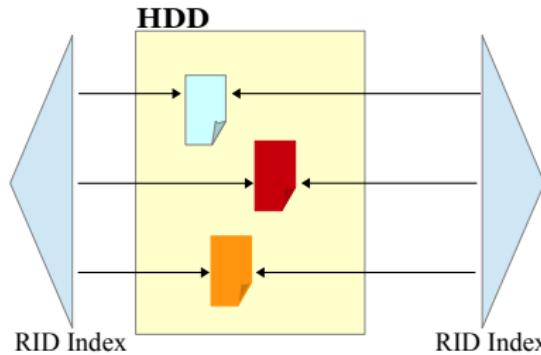
Updating random leaf pages

# Traditional Multi-version Indexing: Updating Records

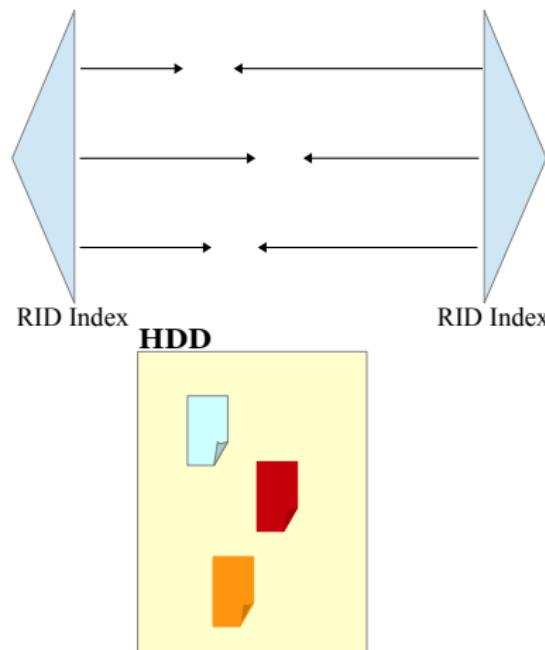


Updating random leaf pages

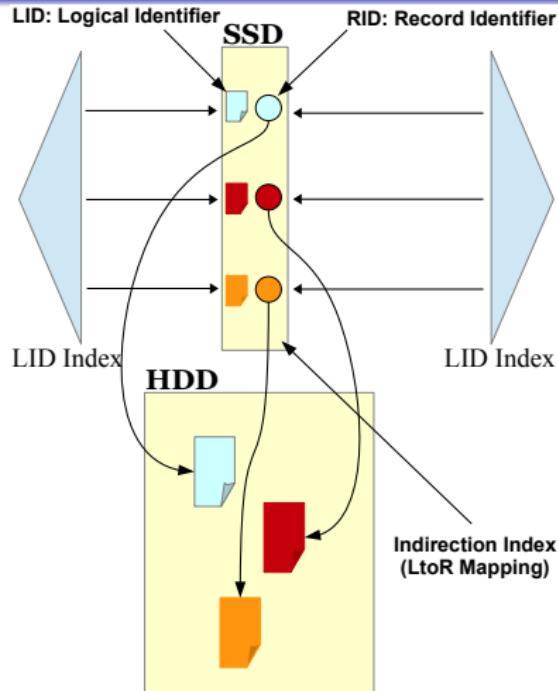
# Indirection Indexing: Updating Records



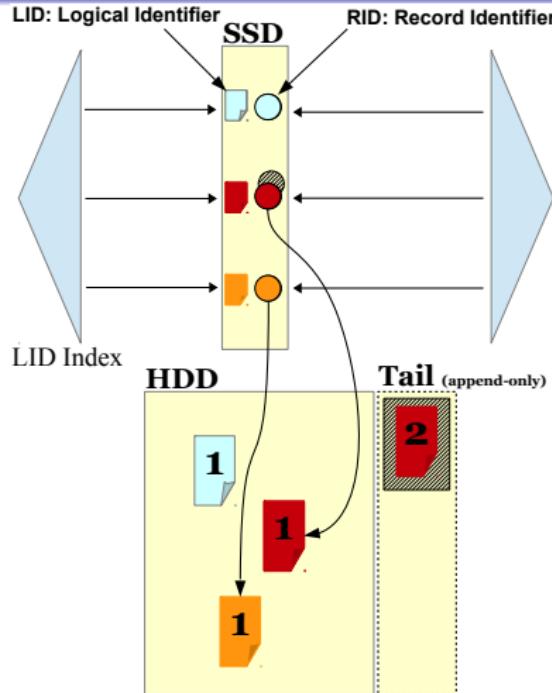
# Indirection Indexing: Updating Records



# Indirection Indexing: Updating Records

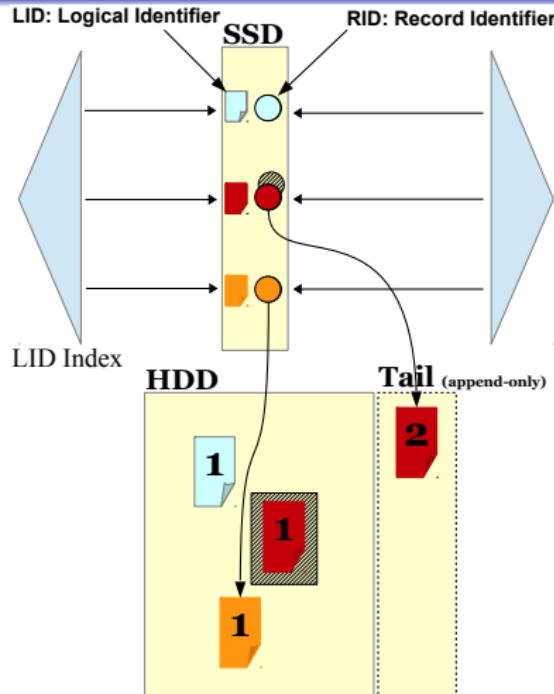


# Indirection Indexing: Updating Records



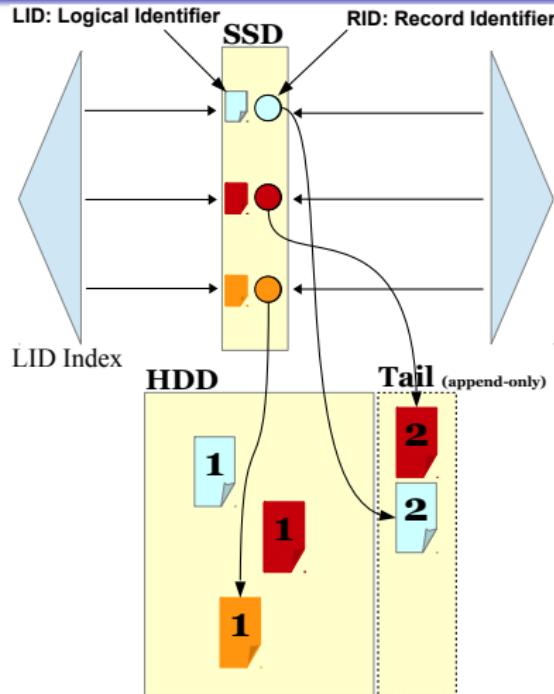
Eliminating random leaf-page updates

# Indirection Indexing: Updating Records



Eliminating random leaf-page updates

# Indirection Indexing: Updating Records



Eliminating random leaf-page updates

# 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

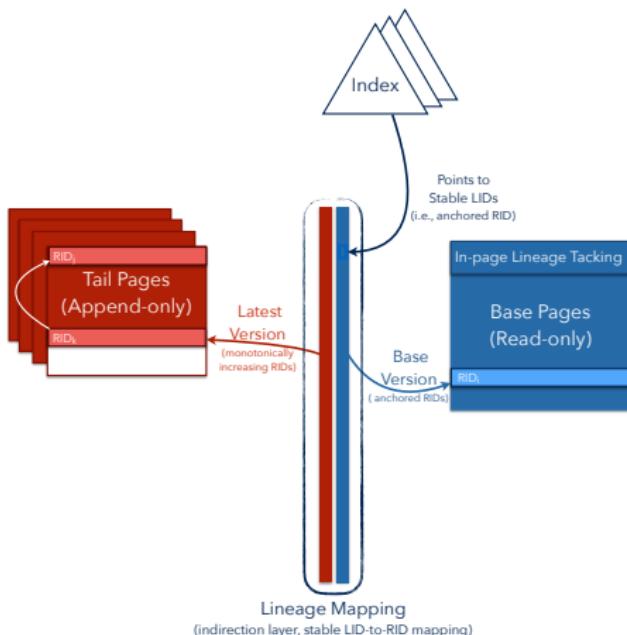
## Observed Trends

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

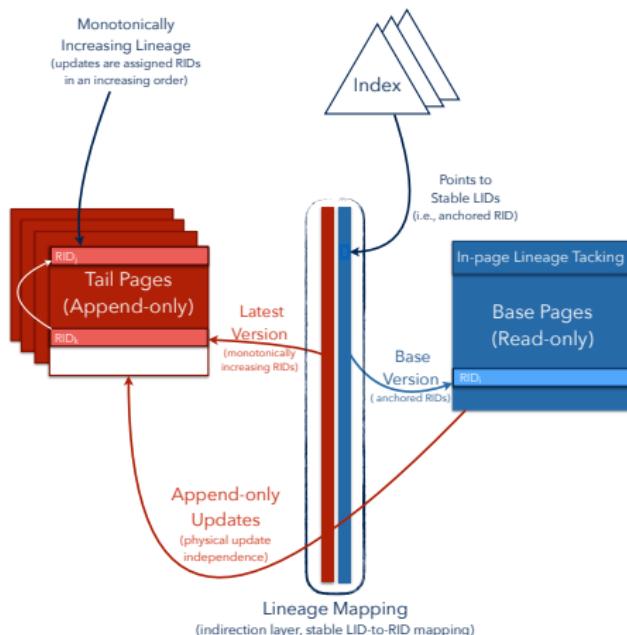
lazily and independently stage stable data from a write-optimized layout (i.e., OLTP) into a read-optimized layout (i.e., OLAP)

# Lineage-based Storage Architecture (LSA): Intuition



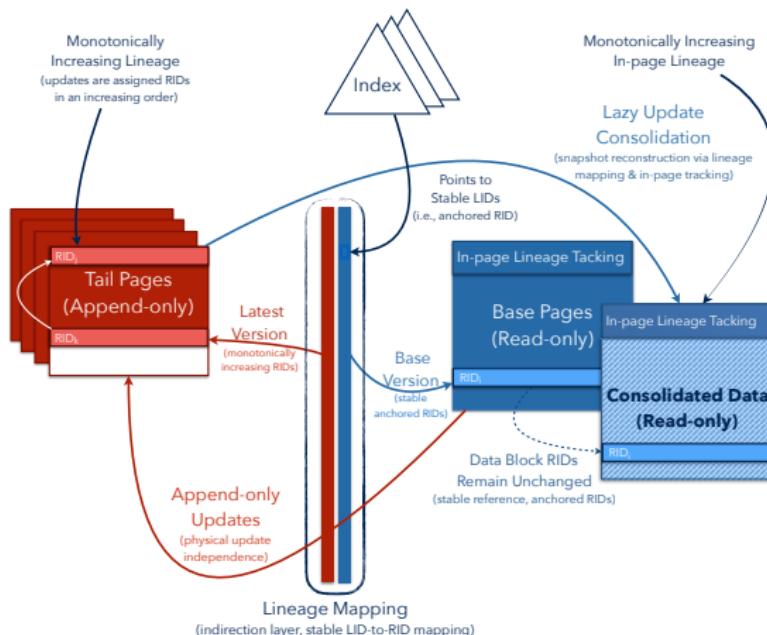
Physical Update Independence: De-coupling data & its updates  
(reconstruction via in-page lineage tracking and lineage mapping)

# Lineage-based Storage Architecture (LSA): Intuition



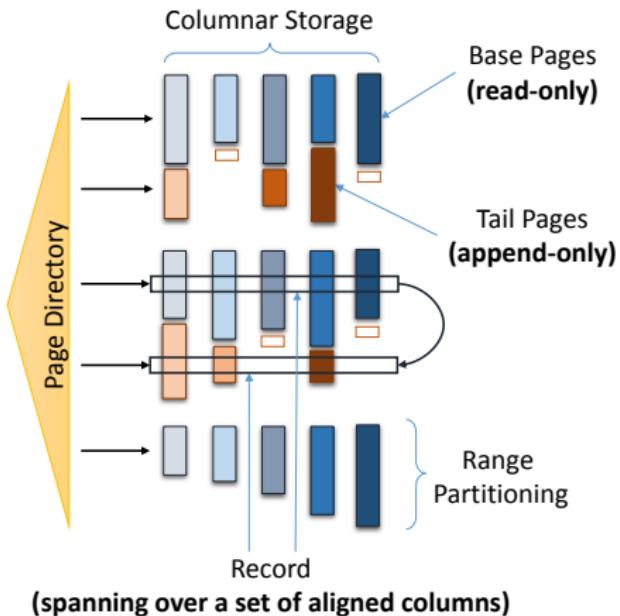
Physical Update Independence: De-coupling data & its updates  
(reconstruction via in-page lineage tracking and lineage mapping)

# Lineage-based Storage Architecture (LSA): Intuition



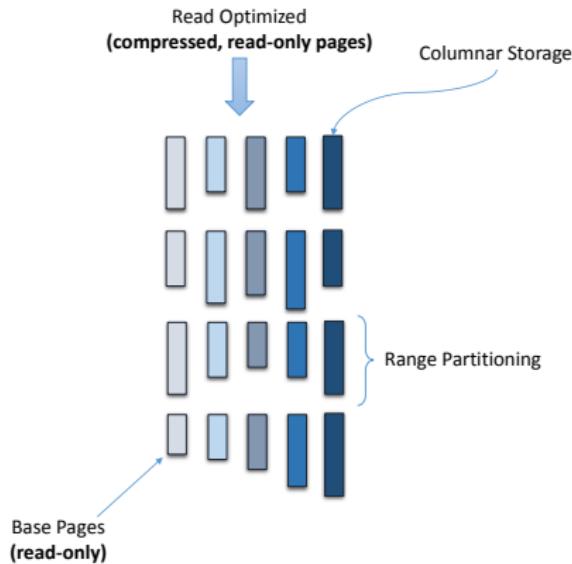
Physical Update Independence: De-coupling data & its updates  
(reconstruction via in-page lineage tracking and lineage mapping)

# Lineage-based Storage Architecture (LSA): Overview



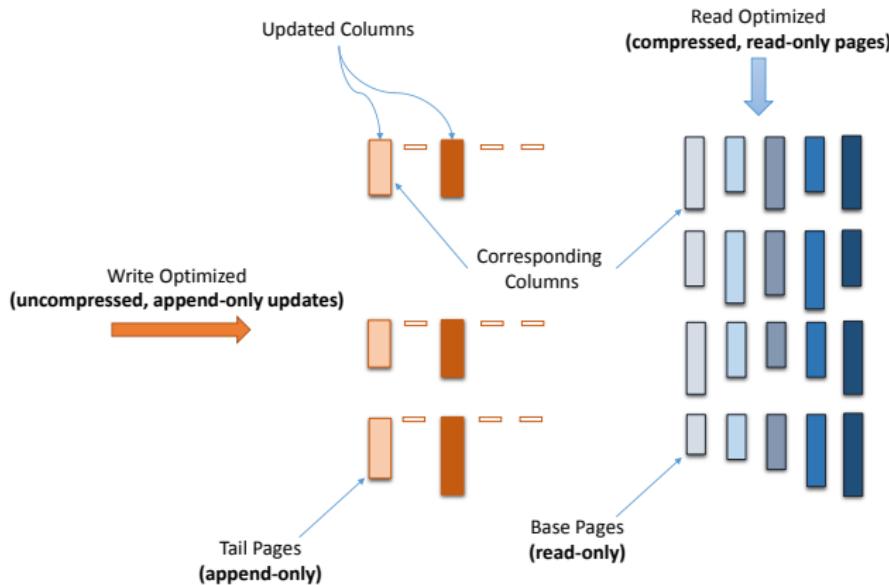
Overview of the lineage-based storage architecture  
**(base pages and tail pages are handled identically at the storage layer)**

# L-Store: Detailed Design



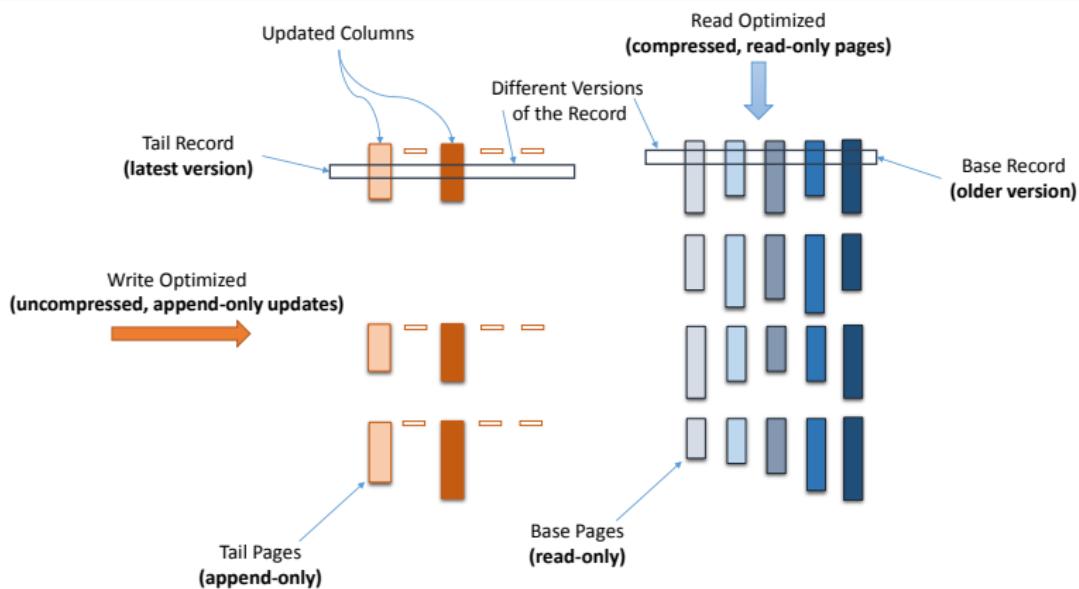
Records are range-partitioned and compressed into a set of ready-only **base pages** (accelerating analytical queries)

# L-Store: Detailed Design



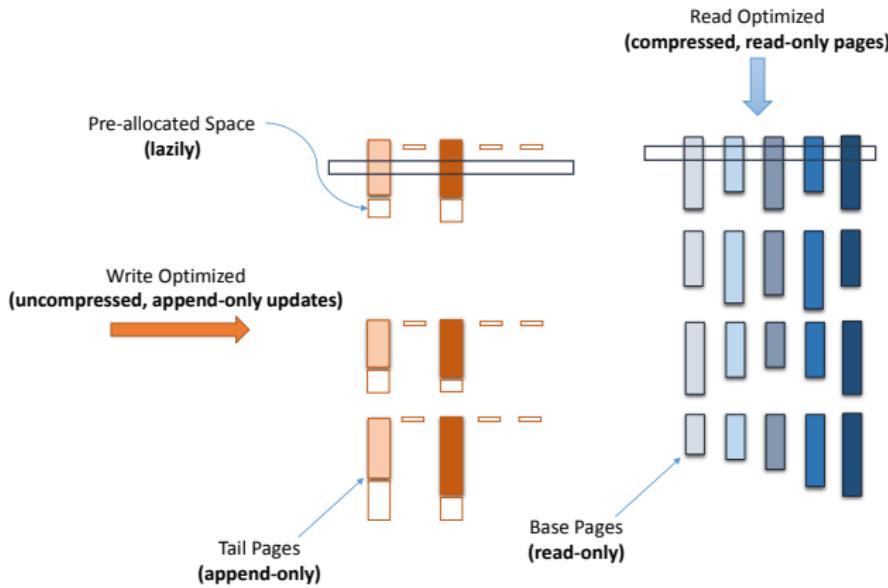
Recent updates for a range of records are clustered in their **tails pages** (transforming costly point updates into an amortized analytical-like query)

# L-Store: Detailed Design



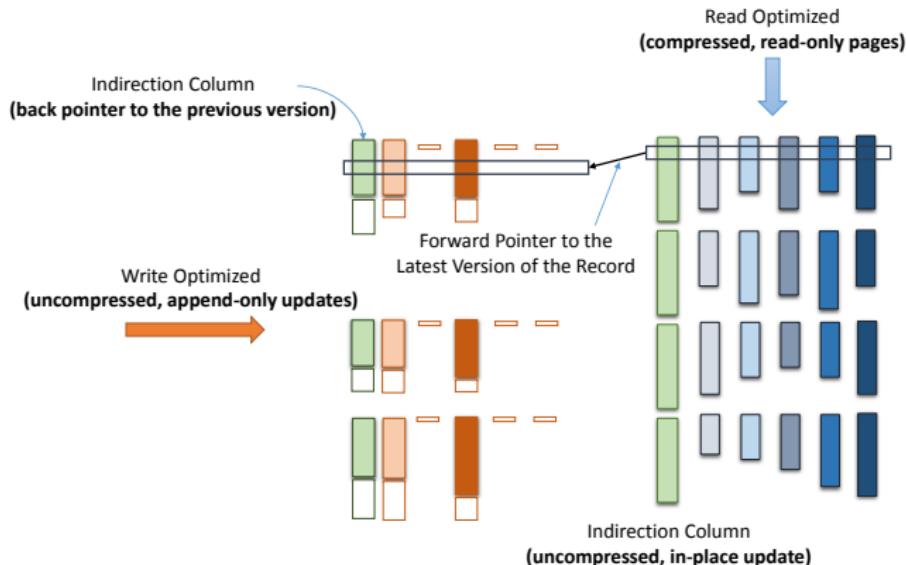
Recent updates for a range of records are clustered in their **tails pages** (transforming costly point updates into an amortized analytical-like query)

# L-Store: Detailed Design



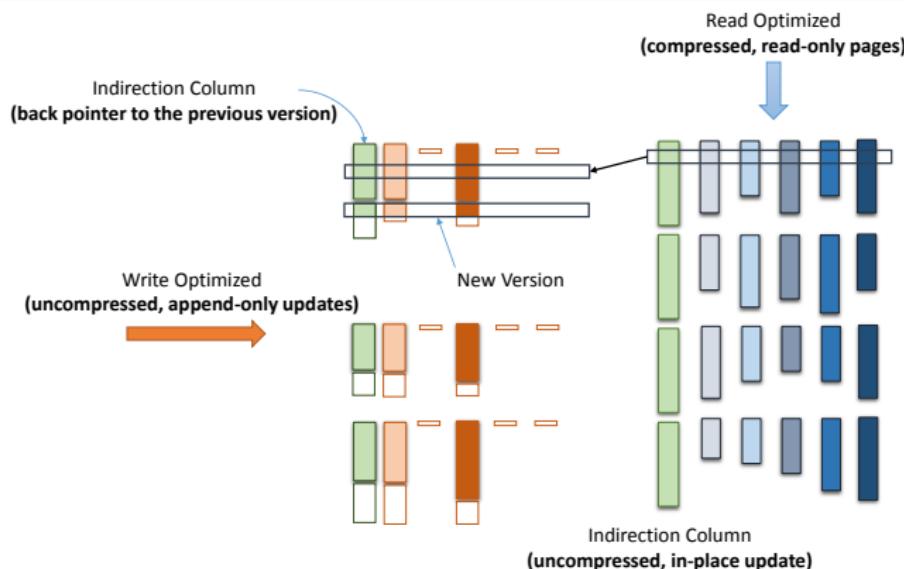
Recent updates are strictly appended, uncompressed in the pre-allocated space  
(eliminating the read/write contention)

# L-Store: Detailed Design



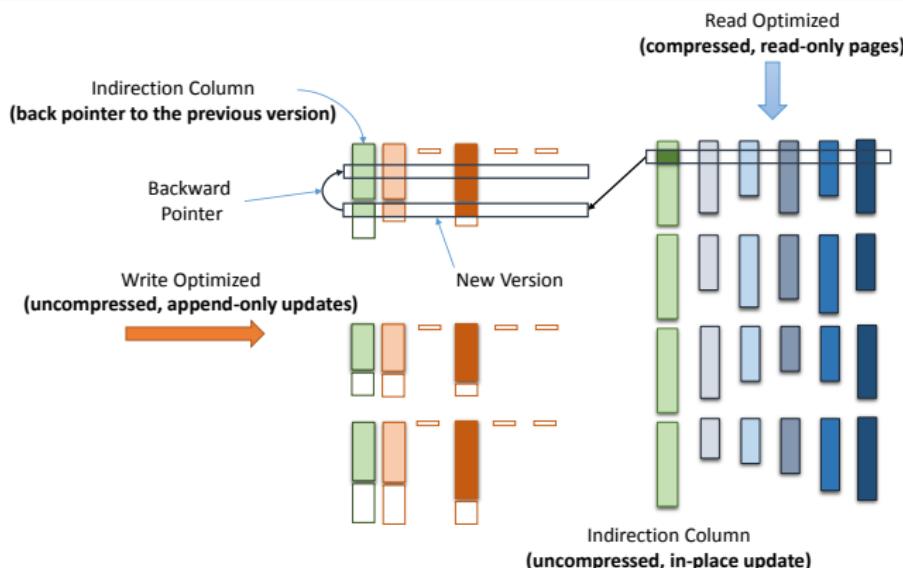
Achieving (at most) 2-hop access to the latest version of any record  
(avoiding read performance deterioration for point queries)

# L-Store: Detailed Design



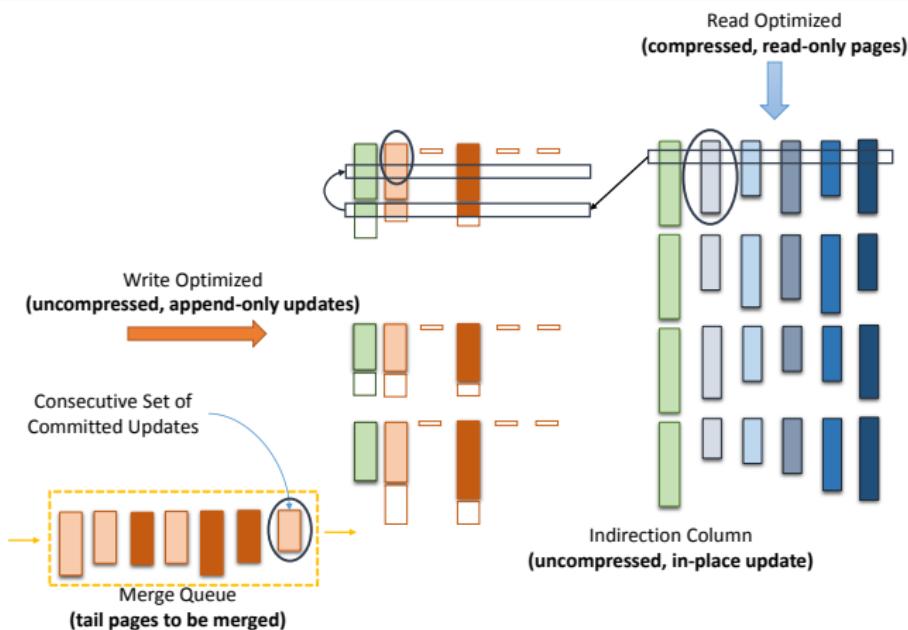
Achieving (at most) 2-hop access to the latest version of any record  
(avoiding read performance deterioration for point queries)

# L-Store: Detailed Design



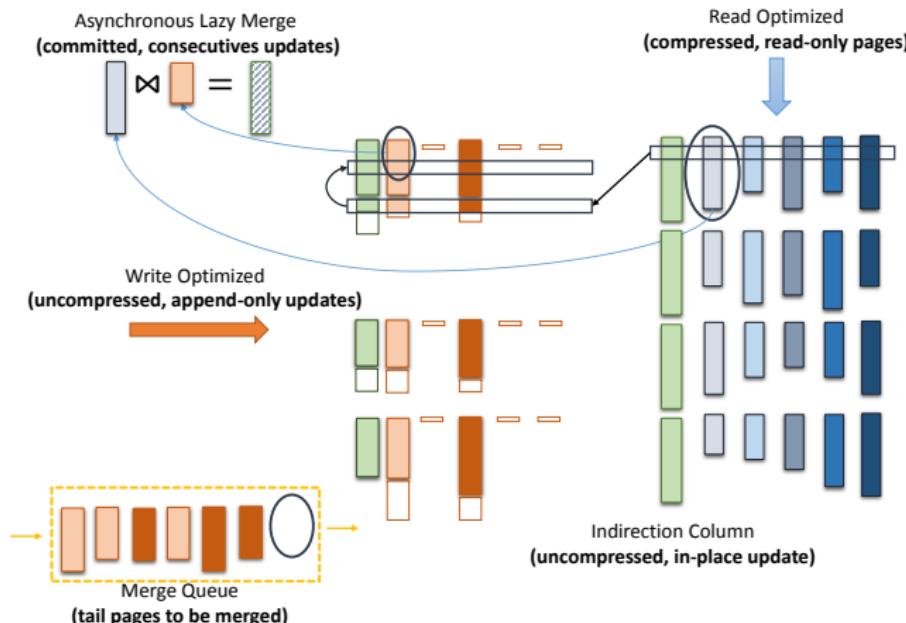
Achieving (at most) 2-hop access to the latest version of any record  
(avoiding read performance deterioration for point queries)

# L-Store: Contention-free Merge



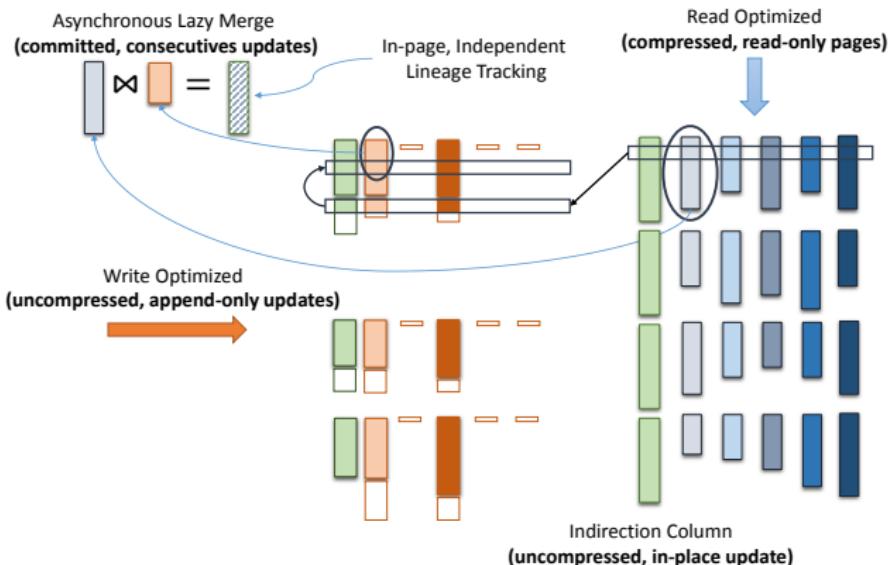
Contention-free merging of only stable data: read-only and committed data  
(no need to block on-going and new transactions)

# L-Store: Contention-free Merge



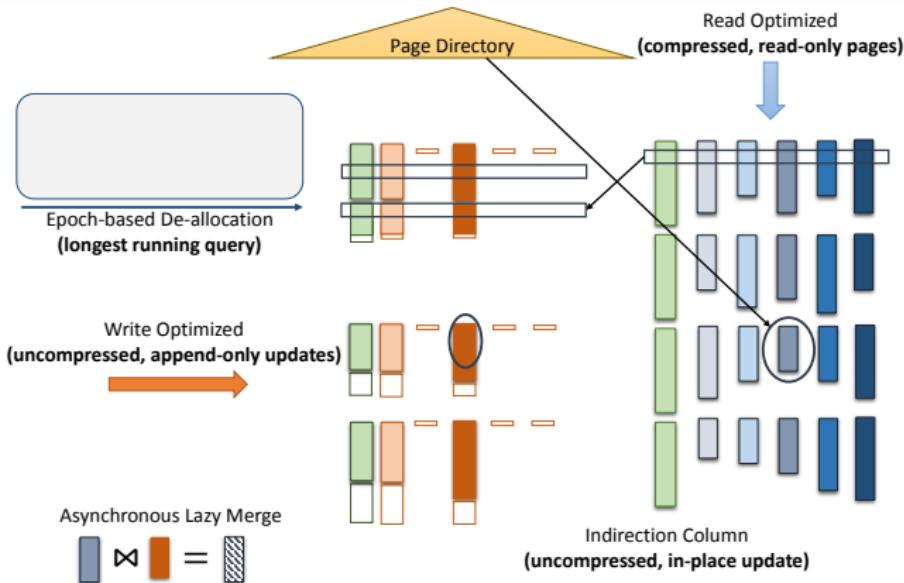
Lazy independent merging of **base pages** with their corresponding **tail pages**  
(resembling a local left outer-join of the base and tail pages)

# L-Store: Contention-free Merge



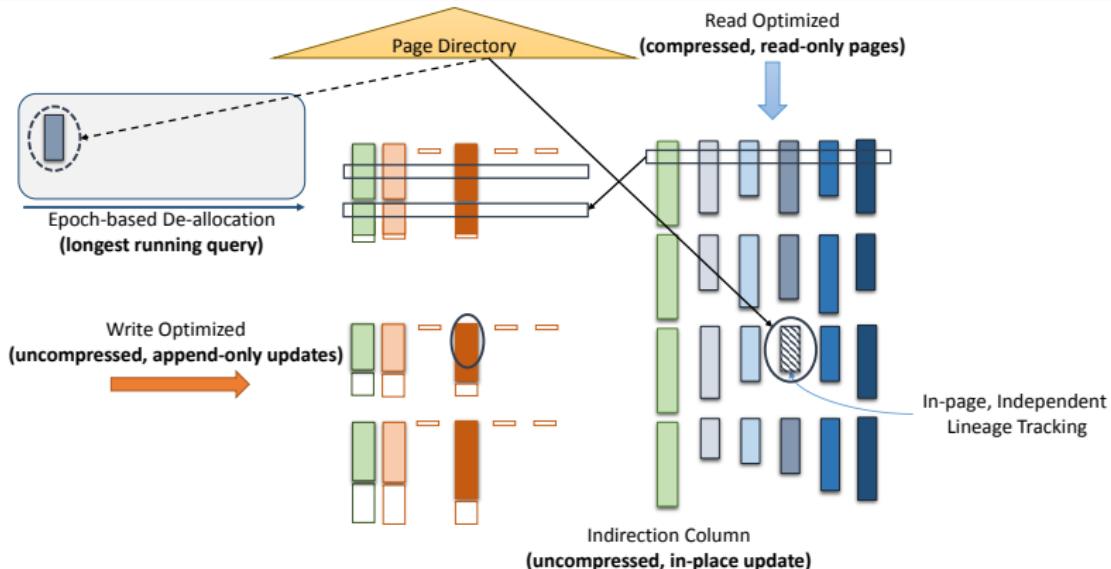
Independently tracking the lineage information within every page  
(no need to coordinate merges among different columns of the same records)

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



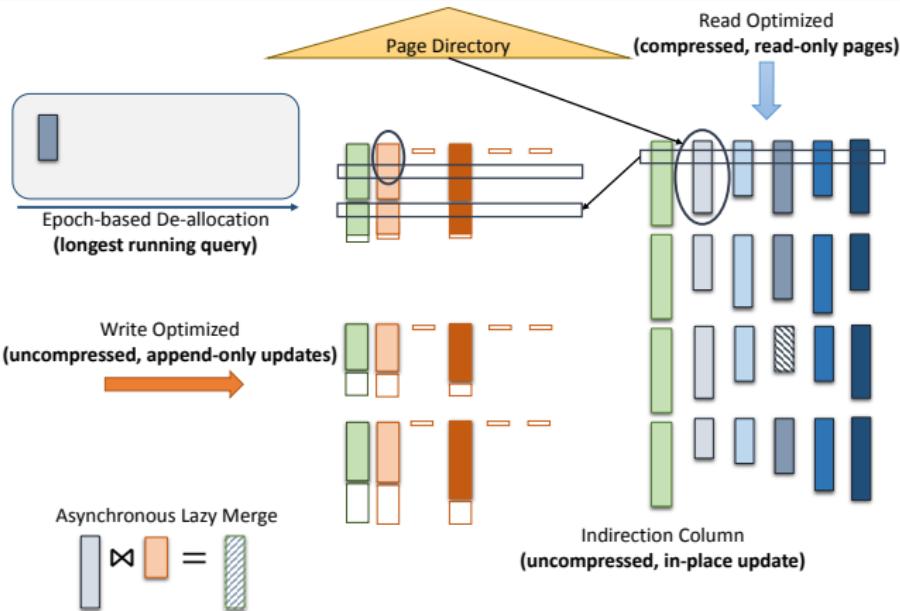
Contention-free page de-allocation using an epoch-based approach  
(no need to drain the ongoing transactions)

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



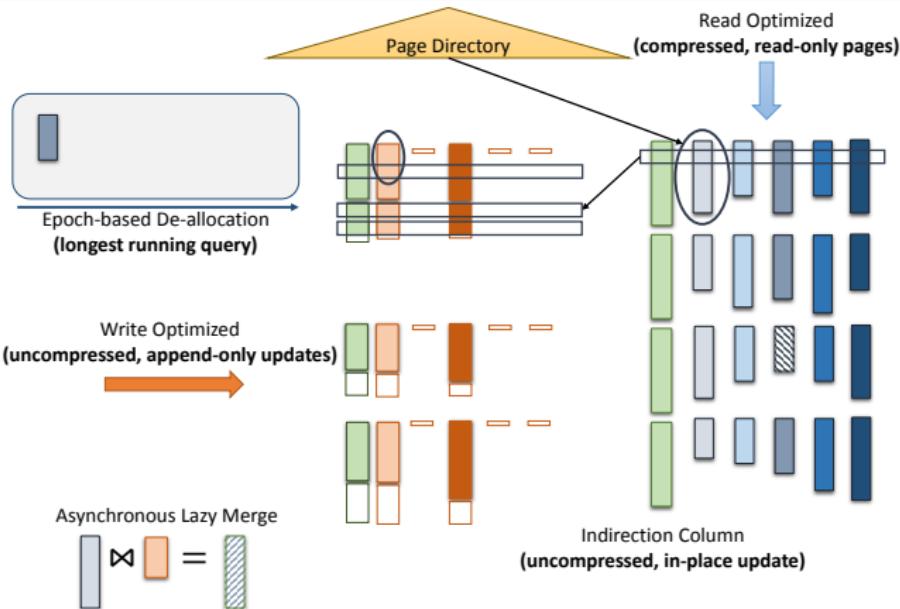
Contention-free page de-allocation using an epoch-based approach  
(no need to drain the ongoing transactions)

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



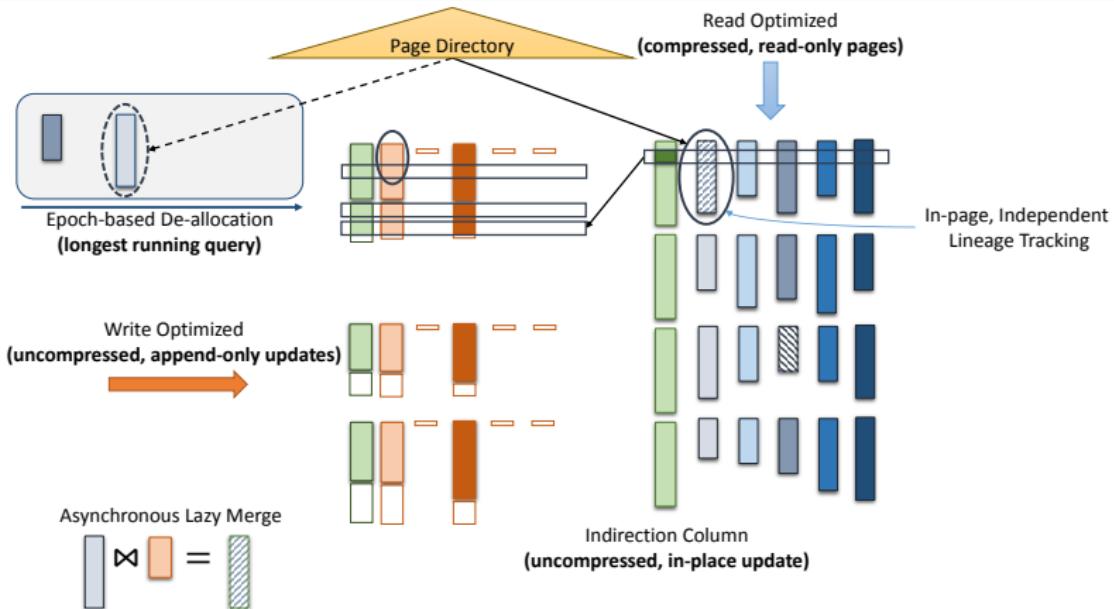
Contention-free page de-allocation using an epoch-based approach  
(no need to drain the ongoing transactions)

# 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



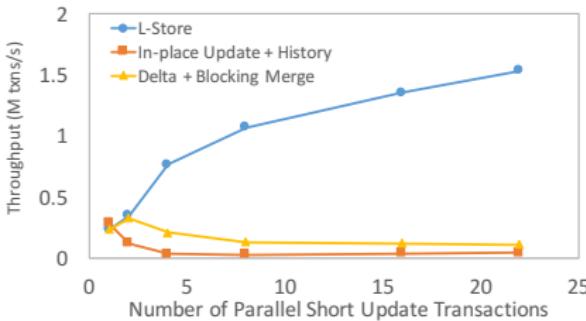
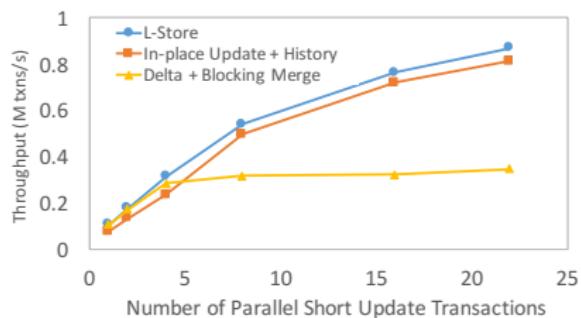
Contention-free page de-allocation using an epoch-based approach  
(no need to drain the ongoing transactions)

# Experimental Analysis

# Experimental Settings

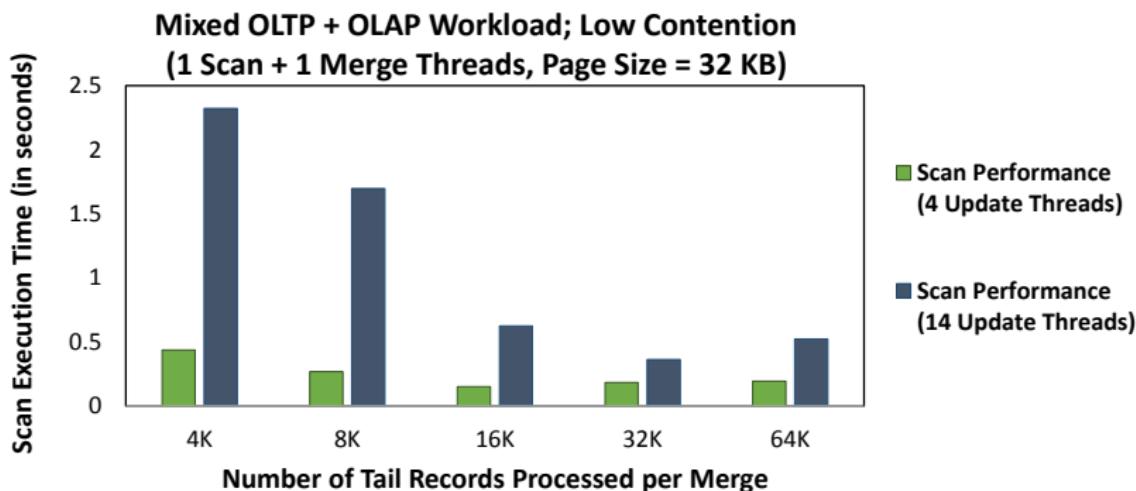
- Hardware:
  - 2 × 6-core Intel(R) Xeon(R) CPU E5-2430 @ 2.20GHz, 64GB, 15 MB L3 cache
- Workload: Extended Microsoft Hekaton Benchmark
  - Comparison with *In-place Update + History* and *Delta + Blocking Merge*
  - Effect of varying contention levels
  - Effect of varying the read/write ratio of short 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



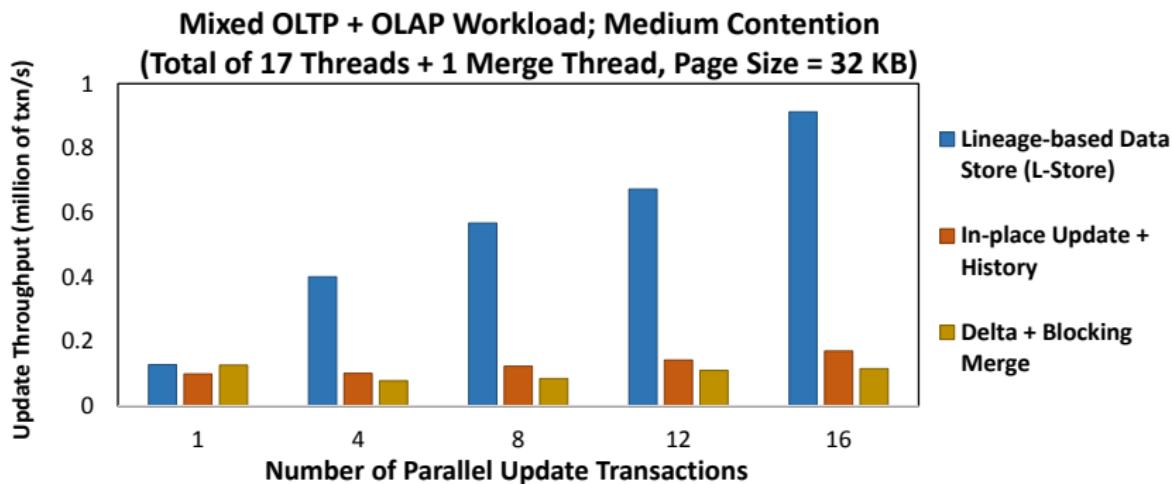
Achieving up to **40×** as increasing the update contention

# Effect of Merge Frequency on Scan Performance



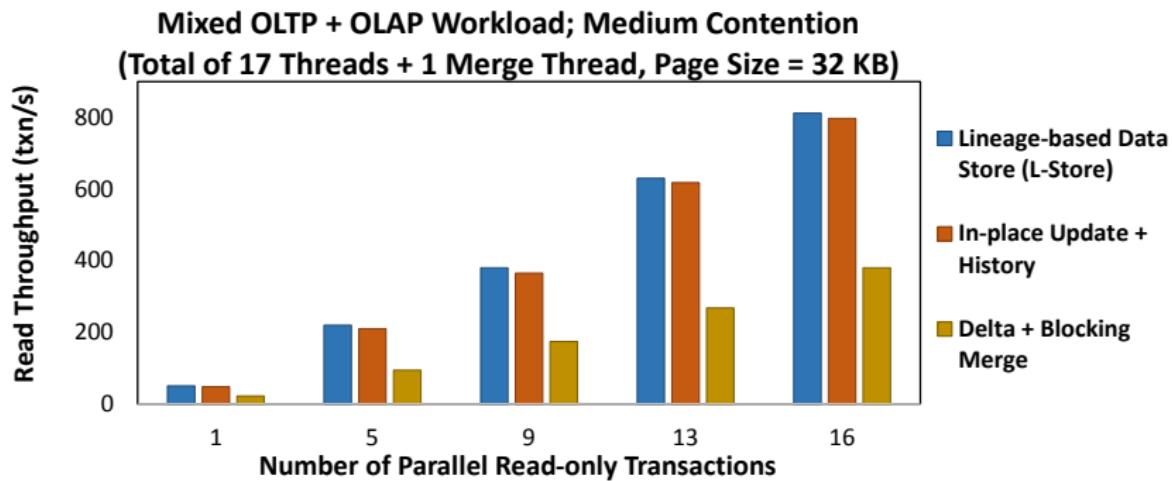
Merge process is essential in maintaining efficient scan performance

# 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://expolab.org/>

