CS 764: Topics in Database Management Systems

Lecture 26: HTAP

Xiangyao Yu

12/2/2020

Project Presentation Schedule

Each team has a 10-min slot: **8-min presentation + 2-min Q/A**

Day 1 (Mon., Dec. 7)	Project Name
1:05 -- 1:15	The SADS Index: Optimizing Search and Insert Operations on a B-Tree Structure
1:15 -- 1:25	Automatic DBMS configuration tuning using Reinforcement Learning techniques
1:25 -- 1:35	One-Phase Commit: A new atomic commit protocol via global log accessibility
1:35 -- 1:45	A Survey on Hybrid Transactional and Analytical processing
1:45 -- 1:55	Survey: Classifying Modern Indexes
1:55 -- 2:05	Empirical Evaluation of Indexing on Modern Database Systems
Day 2 (Wed., Dec. 9)	**Project Name**
1:05 -- 1:15	Efficient updates and inserts with learned indexes
1:15 -- 1:25	Evaluation of Data Compression in GPU Database
1:25 -- 1:35	A survey on recent join algorithms for modern multi-core processor system
1:35 -- 1:45	Comparison of Modern Indexing Approaches on Persistent Memory
1:45 -- 1:55	Data driven techniques for Log Structured Merge Trees
1:55 -- 2:05	Join Optimization with Map Reduce

Today’s Papers: HTAP

F1 Lightning: HTAP as a Service

Jiacheng Yang Ian Rae Jun Xu Jeff Shute Zhan Yuan Kelvin Lau
Qiang Zeng Xi Zhao Jun Ma Ziyang Chen Yuan Gao Qilin Dong
Junxiong Zhou Jeremy Wood Goetz Graefe Jeff Naughton John Cieslewicz
Google LLC
f1-lightning-paper@google.com

ABSTRACT

The ongoing and increasing interest in HTAP (Hybrid Transactional and Analytical Processing) systems documents the intense interest from data owners in simultaneously running transactional and analytical workloads over the same data set. Much of the reported work on HTAP has arisen in the context of “greenfield” systems, answering the question “if we could design a system for HTAP from scratch, what would it look like?” While there is great merit in such an approach, and a lot of valuable technology has been developed with it, we found ourselves facing a different challenge: one in which there is a great deal of transactional data already existing in several transactional systems, heavily queried by an existing federated engine that does not “own” the transactional systems, supporting both new and legacy applications that demand transparent fast queries and

Simply put, while supporting HTAP well is of critical importance, for us a greenfield approach was not the best option to enable HTAP processing in Google’s ecosystem. In Google, we use multiple transactional data stores that serve large legacy and new workloads, and we have federated query engines that are loosely coupled with these systems. We want a single HTAP solution that can be enabled across the different options for transactional storage to avoid costly migrations and to permit flexibility in the design of transactional storage systems, and we want to benefit from separation of concerns by allowing transactional systems to focus on transaction processing and query engines to focus on query processing, with an emphasis on analytical queries.

Accordingly, we have designed, implemented, and deployed Lightning, a loosely coupled HTAP solution that we term “HTAP-as-a-service.” By “HTAP-as-a-service” we mean that Lightning can

VLDB 2020

HTAP: Hybrid Transactional/Analytical Processing

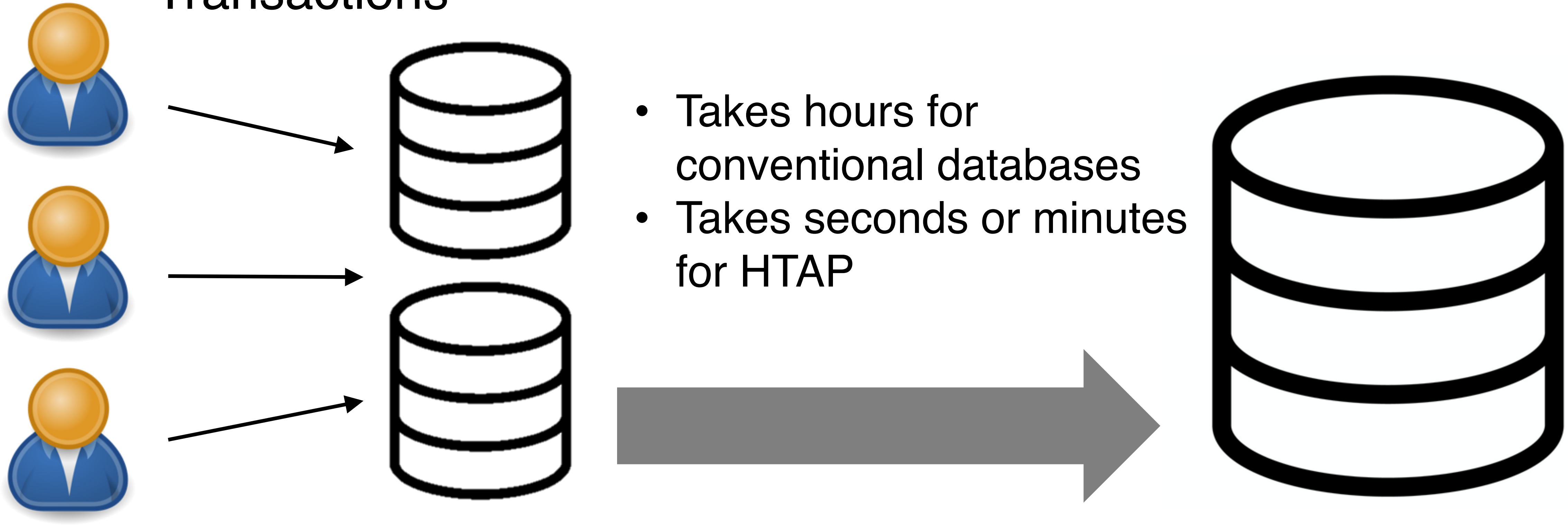
Hybrid transactional/analytical processing (HTAP), a term created by Gartner Inc in 2014:

> Hybrid transactional/analytical processing (HTAP) is an emerging application architecture that "breaks the wall" between transaction processing and analytics. It enables more informed and "in business real time" decision making.

Key advantage: **reducing time to insight**

OLTP vs. OLAP

Transactions



- Takes hours for conventional databases
- Takes seconds or minutes for HTAP

OLTP database

(Update Intensive)

OLAP database

(Read Intensive, rare updates)

HTAP Design Options [1]

Single System for OLTP and OLAP

- *Using Separate Data Organization for OLTP and OLAP*
- *Same Data Organization for both OLTP and OLAP*

Separate OLTP and OLAP Systems

- *Decoupling the Storage for OLTP and OLAP*
- *Using the Same Storage for OLTP and OLAP*

[1] Özcan, Fatma, Yuanyuan Tian, and Pinar Tözün. "Hybrid transactional/analytical processing: A survey." *ICMD,* 2017.

HTAP Design Options [1]

Single System for OLTP and OLAP

- *Using Separate Data Organization for OLTP and OLAP*
- *Same Data Organization for both OLTP and OLAP*

Separate OLTP and OLAP Systems

- *Decoupling the Storage for OLTP and OLAP*
- *Using the Same Storage for OLTP and OLAP*

F1 Lightning

[1] Özcan, Fatma, Yuanyuan Tian, and Pinar Tözün. "Hybrid transactional/analytical processing: A survey." *ICMD,* 2017.

Benefits of Separating OLTP and OLAP

Examples: F1 lightning, TiDB, SAP HANA, Oracle database (partially)

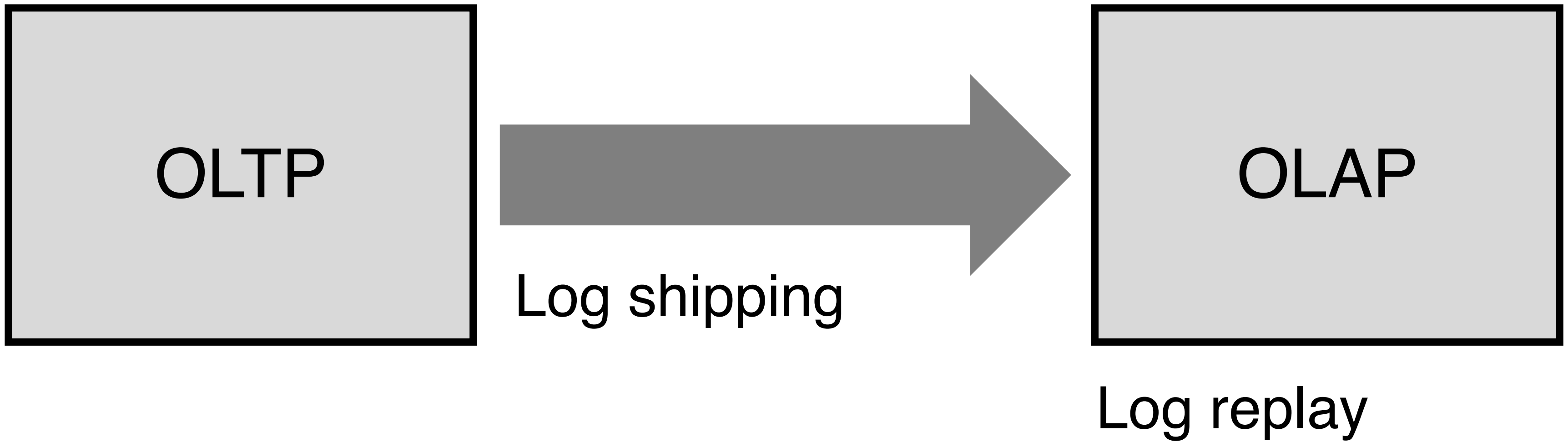
Separation of concerns

- The OLTP and OLAP may be implemented and used by different teams

Independent performance optimizations

Compatible with existing OLTP services

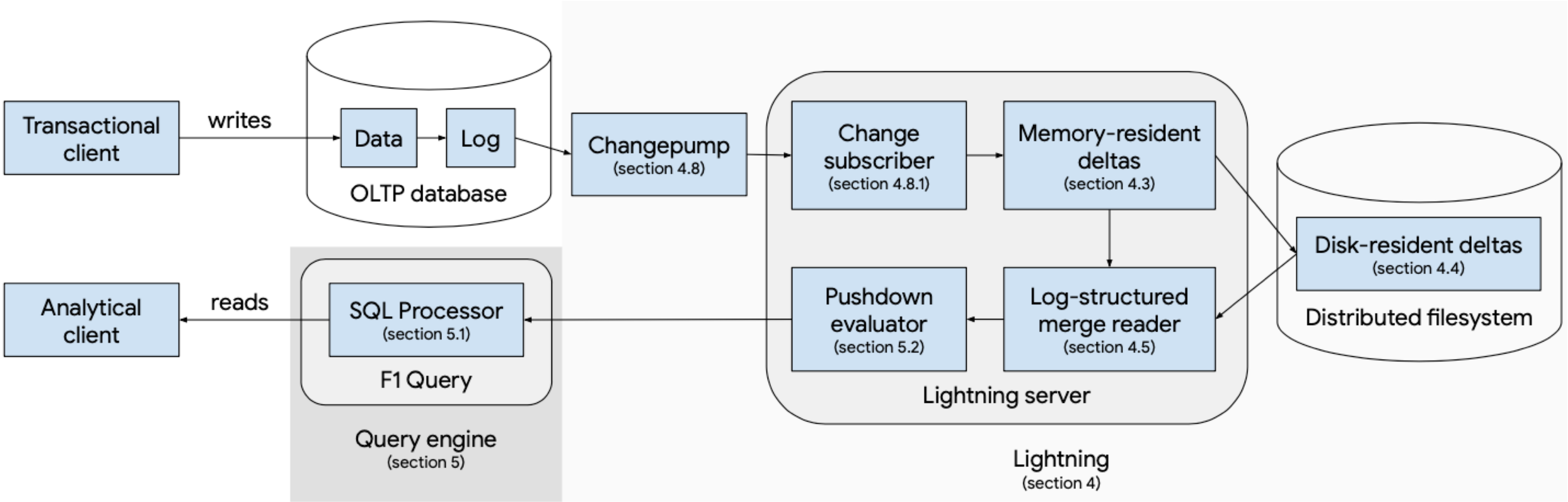
HTAP with Separate OLTP and OLAP



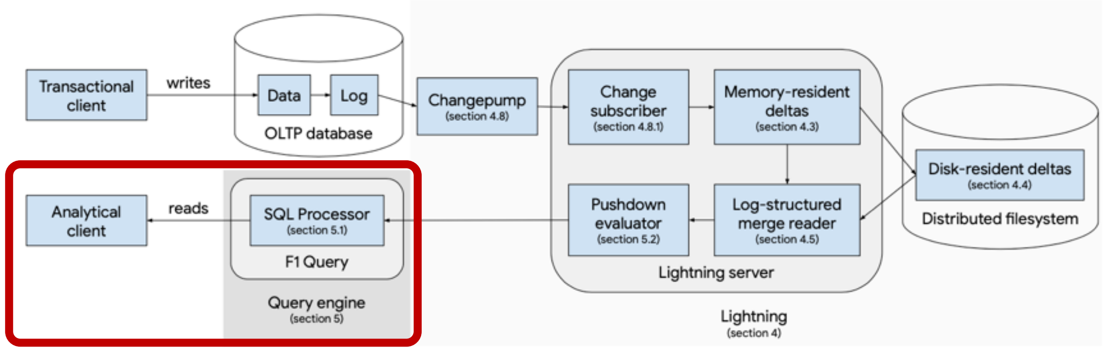
Important metrics:

- OLTP and OLAP **throughput** and **latency**
- **Interference** between the two engines
- **Freshness** of OLAP queries

F1 Lightning Architecture



Read Semantics



MVCC with snapshot isolation

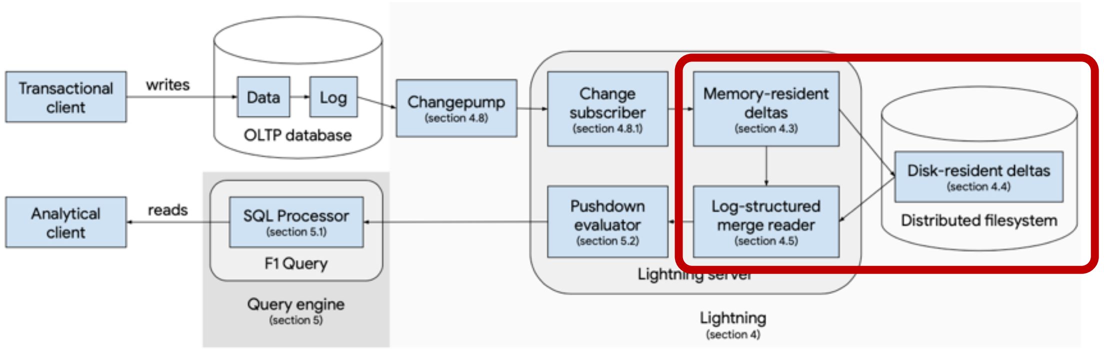
Queryable window

- Maximum safe timestamp
- Minimum safe timestamp
- Typical queryable window is 10 hours

Tables and Deltas

Delta: partial row versions

- Insert: all columns
- Update: modified columns
- Delete: no column value



Tables and Deltas

Delta: partial row versions

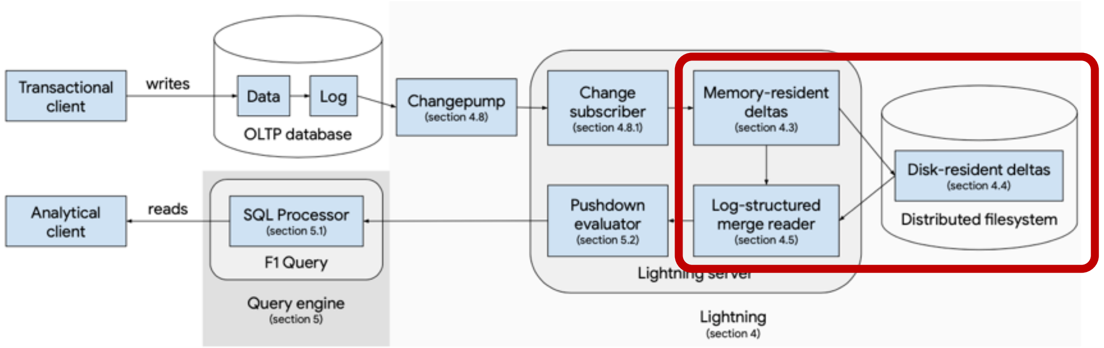
- Insert: all columns
- Update: modified columns
- Delete: no column value

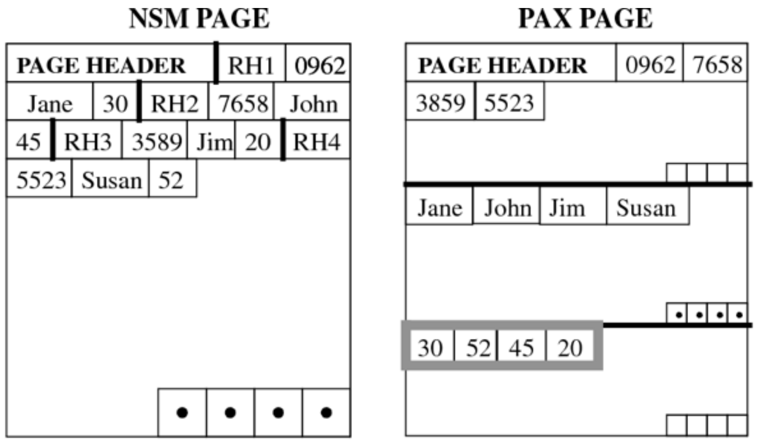
Memory resident deltas

- Row store B-tree

Disk resident deltas

- Data part: PAX (Partition Attributes Across) format
- Index part: sparse B-tree on the primary keys





Tables and Deltas

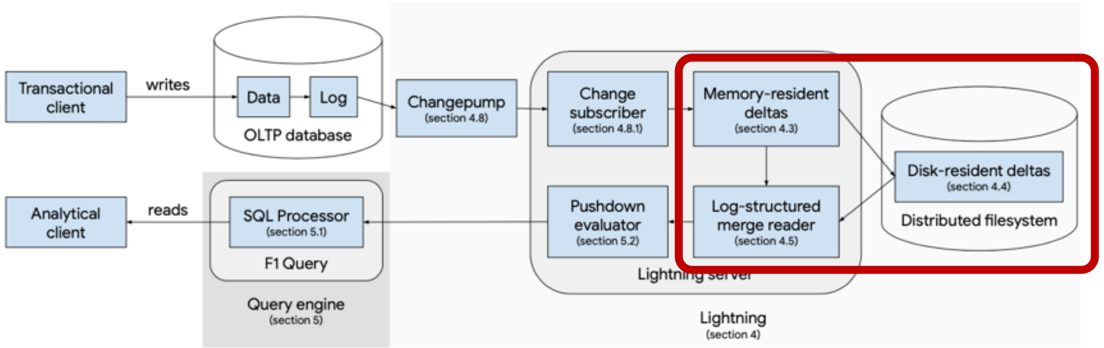
Delta: partial row versions
- Insert: all columns
- Update: modified columns
- Delete: no column value

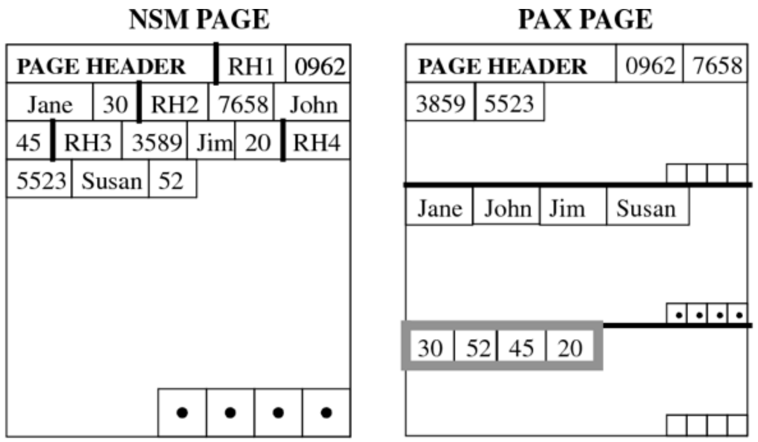
Memory resident deltas
- Row store B-tree

Disk resident deltas
- Data part: PAX (Partition Attributes Across) format
- Index part: sparse B-tree on the primary keys

Delta merging

Delta compaction
- Rewrite smaller deltas into a single large delta





Schema Management

Id (INT)	TS (INT)	OP (ENUM)	Name (STRING)	Address (STRUCT)
1	150	UPDATE	*NotSet*	{city: “Madison” state: “WI”}
1	125	UPDATE	*NotSet*	{city: “Milwaukee” state: “WI”}
1	100	INSERT	John Smith	{city: “Seattle” state: “WA”}
2	50	INSERT	Jane Doe	{city: “San Jose” state: “CA”}

(a) Partial row versions conforming to a logical schema.

Logical column	Physical column
Id	Id
TS	TS
OP	OP
Address	Address
Address.City	City
Address.State	State

(b) A mapping between logical and physical columns.

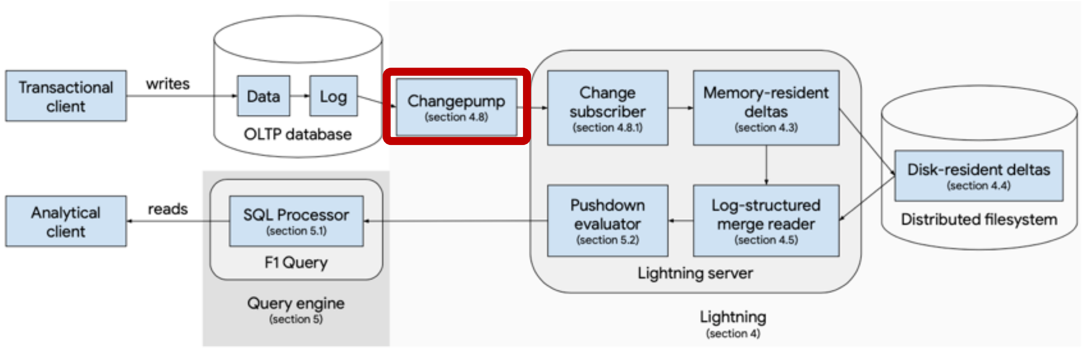
Id (INT)	TS (INT)	OP (INT)	Name (STRING)	Address (STRING)	City (STRING)	State (STRING)
1	150	UPDATE	*NotSet*	{city: “Madison” state: “WI”}	Madison	*NotSet*
1	125	UPDATE	*NotSet*	{city: “Milwaukee” state: “WI”}	Milwaukee	WI
1	100	INSERT	John Smith	{city: “Seattle” state: “WA”}	Seattle	WA
2	50	INSERT	Jane Doe	{city: “San Jose” state: “CA”}	San Jose	CA

(c) Partial row versions conforming to a physical schema.

Benefits of separating logical and physical schemas

- Allows alternative storage layouts for the same logical data
- Facilitates metadata-only schema changes (e.g., adding and dropping a column)

Change Pump



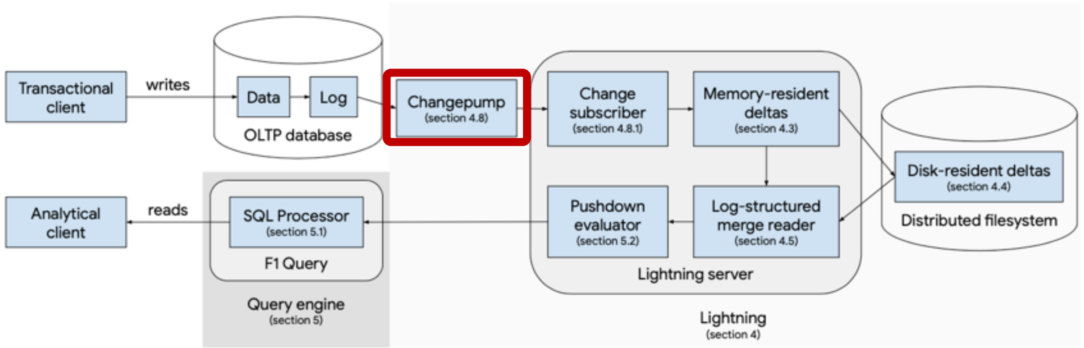
Subscription

- A lightning server subscribes to Changepump with table and key range

Change data

- **Checkpoint timestamp**: changes prior to it have been delivered (use it to update max safe timestamp)

Change Pump



Subscription

- A lightning server subscribes to Changepump with table and key range

Change data

- **Checkpoint timestamp**: changes prior to it have been delivered (use it to update max safe timestamp)

Sharding

- Requires **Shuffle**: Changepump and Lightning partition strategy can be different

Caching

- Speedup data replication across replicas of Lightning servers
- Speedup recovery if a Lightning server fails

Fault Tolerance

Query failures

- Replicate Lightning servers: they can all serve queries

Fault Tolerance

Query failures

- Replicate Lightning servers: they can all serve queries

Ingestion failures

- Changepump server crash -> Replicate Changepump servers
- Outage of OLTP system -> Switch to a healthy datacenter when slowness is detected

Fault Tolerance

Query failures

- Replicate Lightning servers: they can all serve queries

Ingestion failures

- Changepump server crash -> Replicate Changepump servers
- Outage of OLTP system -> Switch to a healthy datacenter when slowness is detected

Table-level failures

- Queries to blacklisted tables are served by the OLTP system
- Case 1: data corruption
- Case 2: lightning cannot keep up with the log (e.g., high rate of change)

Evaluation – Freshness

Queries read data that is 7–12 min stale

Research question: **how to improve freshness of queries**?

Evaluation – CPU Efficiency Improvement

	Small	Medium	Large
Data source CPU time speed-up	2.3x	11.8x	7.6x
F1 server CPU time speed-up	1.5x	16.9x	3.8x

Lightning is faster and more efficient than the OLTP engines (e.g., F1 DB)

Hyper

HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots

Alfons Kemper[1], Thomas Neumann[2]

Fakultät für Informatik
Technische Universität München
Boltzmannstraße 3, D-85748 Garching
[1]kemper@in.tum.de
[2]neumann@in.tum.de

***Abstract*—The two areas of online transaction processing (OLTP) and online analytical processing (OLAP) present different challenges for database architectures. Currently, customers with high rates of mission-critical transactions have split their data into two separate systems, one database for OLTP and one so-called *data warehouse* for OLAP. While allowing for decent transaction rates, this separation has many disadvantages including data freshness issues due to the delay caused by only pe-**

database system. In addition, a separate Data Warehouse system is installed for business intelligence query processing. Periodically, e.g., during the night, the OLTP database changes are extracted, transformed to the layout of the data warehouse schema, and loaded into the data warehouse. This data staging and its associated ETL (Extract–Transform–Load) obviously incurs the problem of *data staleness* as the ETL process can

ICDE 2011

HTAP Design Options [1]

Single System for OLTP and OLAP

- *Using Separate Data Organization for OLTP and OLAP*
- *Same Data Organization for both OLTP and OLAP*

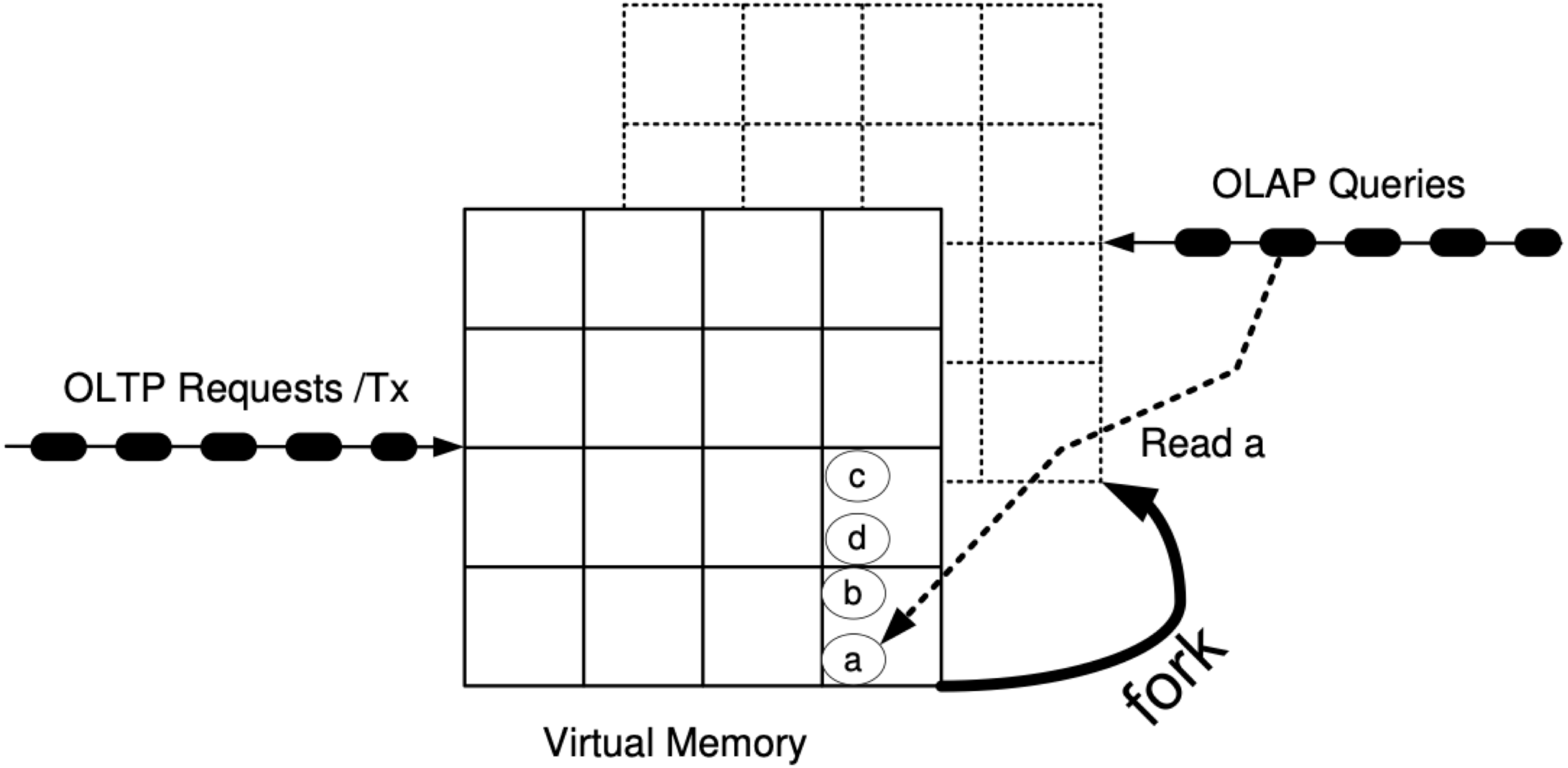
Hyper

Separate OLTP and OLAP Systems

- *Decoupling the Storage for OLTP and OLAP*
- *Using the Same Storage for OLTP and OLAP*

[1] Özcan, Fatma, Yuanyuan Tian, and Pinar Tözün. "Hybrid transactional/analytical processing: A survey." *ICMD,* 2017.

Virtual Memory Snapshots



Create consistent database snapshot for OLAP queries to read

Transactions run with copy-on-write to avoid polluting the snapshots

Fork()

Linux Programmer's Manual

> **fork**() creates a new process by duplicating the calling process. The new process is referred to as the *child* process. The calling process is referred to as the *parent* process.

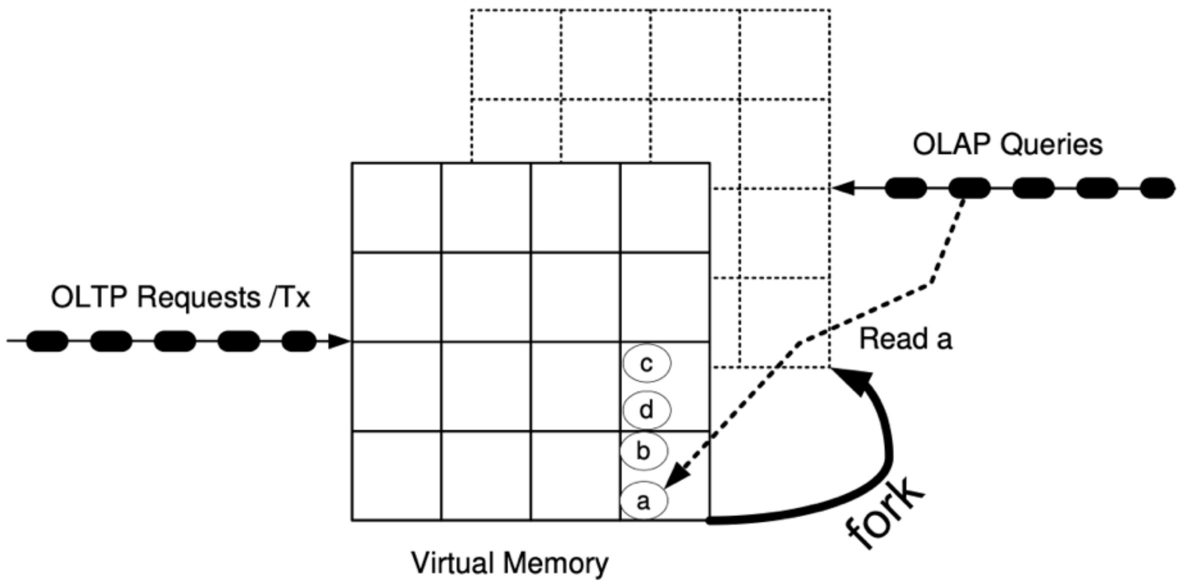
Does **not** copy all the memory pages

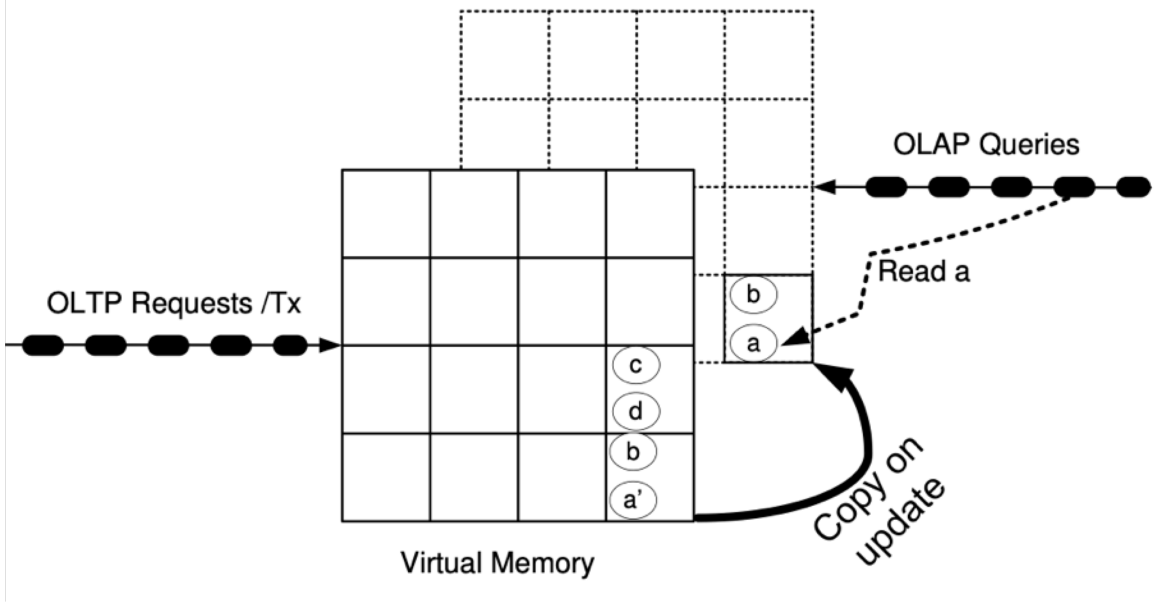
Does copy the parent’s page table (all pages set to readonly mode)

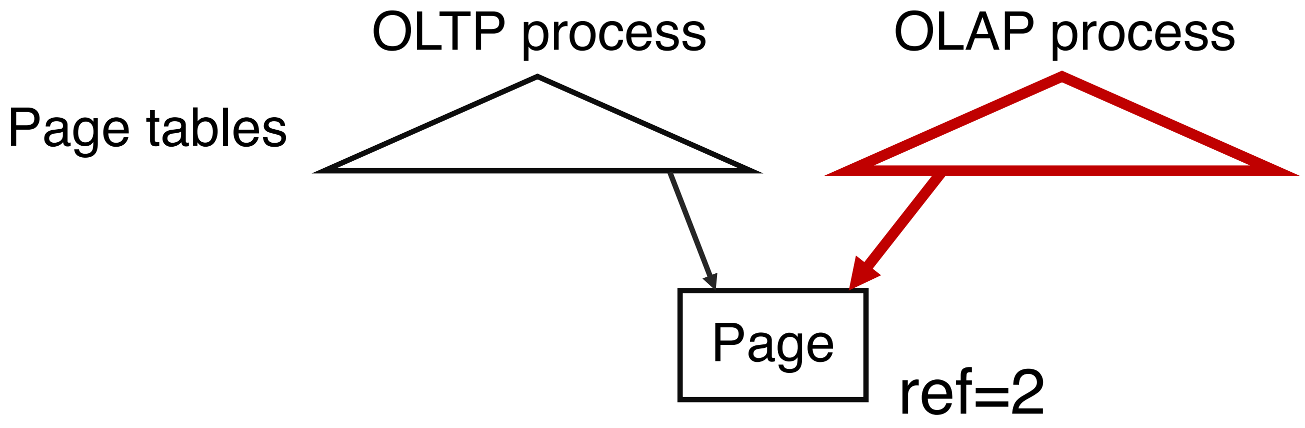
Copy-on-write (COW)

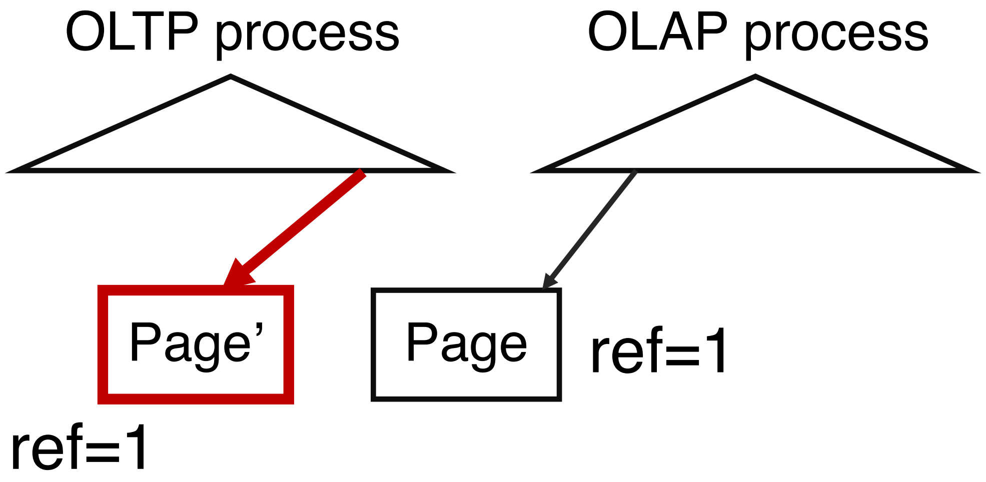
- If any page is modified by either parent or child process, a new page is created for the corresponding process

Fork-Based Virtual Snapshots

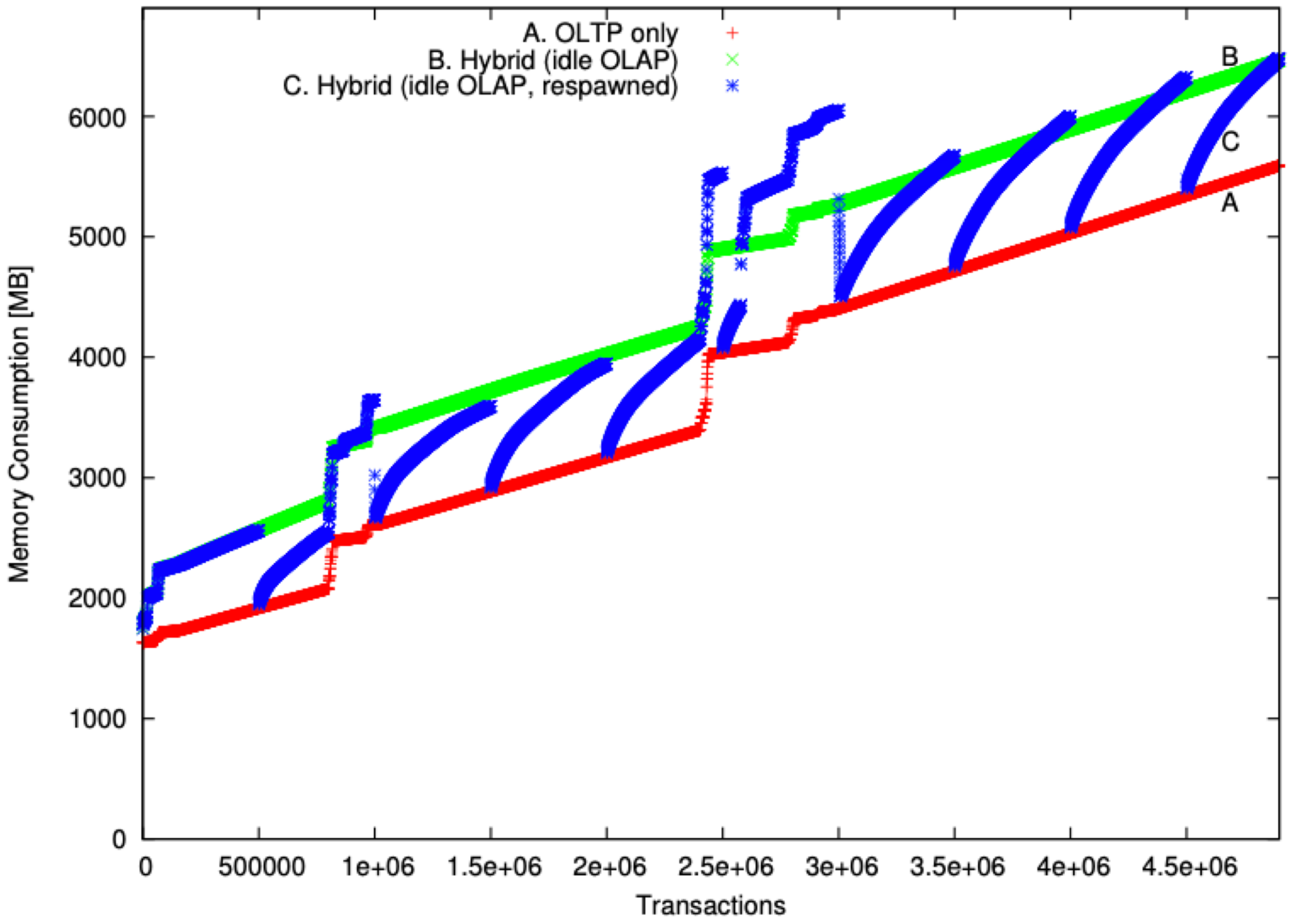








Evaluation – Memory Consumption



Summary

Separating OLTP and OLAP

Examples: F1 lightning, TiDB, SAP HANA, Oracle database (partially)

Advantages:

- Separation of concerns
- Independent performance optimizations
- Compatible with existing OLTP services

Unified storage for OLTP and OLAP

Examples: Hyper, SingleStore, Greenplum, MySQL, PostgreSQL

Advantages:

Q/A – HTAP

Paper contains huge amount of information

Which type of HTAP system is more popular?

How does Lightning compare to greenfield systems in performance?

The paper has little evaluation