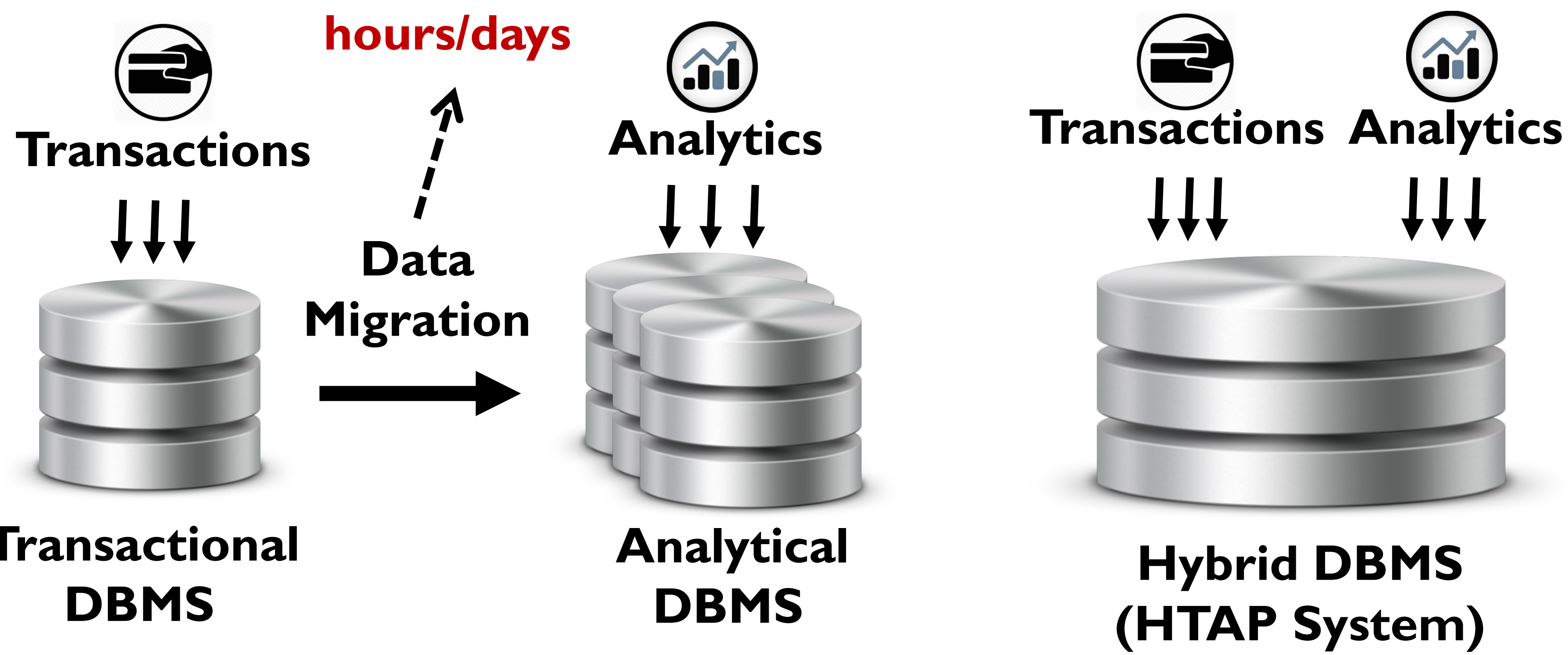
Polynesia:
Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand
Geraldo F. Oliveira
Saugata Ghose
Onur Mutlu

ICDE
2022

HTAP: Supporting Real-Time Analysis

Traditionally, new transactions (updates) are propagated to the analytical database using a periodic and costly process



To support real-time analysis: a single hybrid DBMS is used to execute both transactional and analytical workloads

Ideal HTAP System Properties

An ideal HTAP system should have three properties:

1 **Workload-Specific Optimizations**

- Transactional and analytical workloads must benefit from their own specific optimizations

2 **Data Freshness and Consistency Guarantees**

- Guarantee access to the most recent version of data for analytics while ensuring that transactional and analytical workloads have a consistent view of data

3 **Performance Isolation**

- Latency and throughput of transactional and analytical workloads are the same as if they were run in isolation

Achieving all three properties at the same time is very challenging

Problem and Goal

Problems:

1 **State-of-the-art HTAP systems do not achieve all of the desired HTAP properties**

2 **Data freshness and consistency mechanisms are data-intensive and cause a drastic reduction in throughput**

3 **These systems fail to provide performance isolation because of high resource contention**

Goal:

4 **Take advantage of custom algorithm and processing-in-memory (PIM) to address these challenges**

Polynesia

Key idea: **partition computing resources into two types of isolated and specialized processing islands**

Isolating transactional islands from analytical islands allows us to:

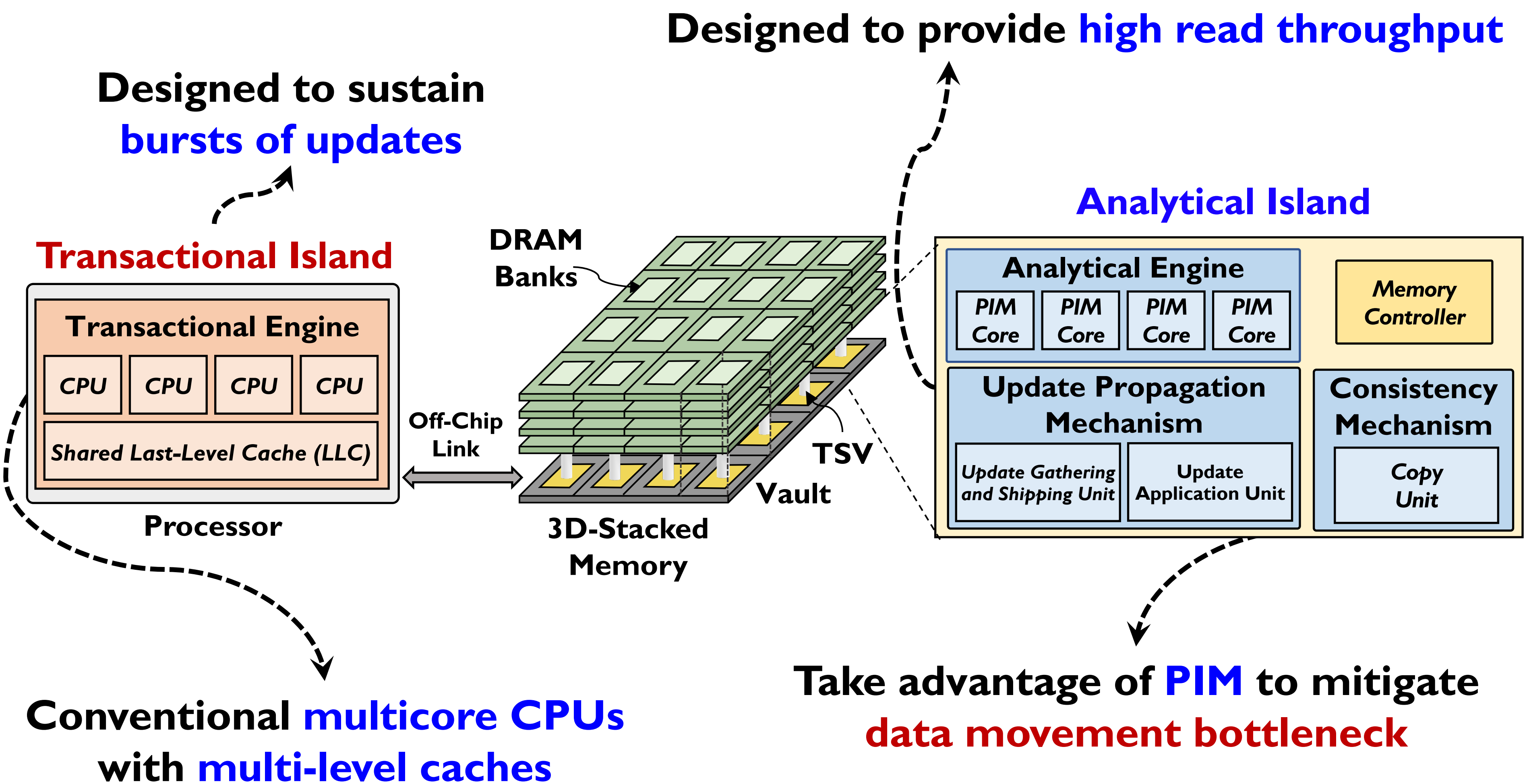
1. **Apply workload-specific optimizations to each island**

2. **Avoid high resource contention**

3. **Design efficient data freshness and consistency mechanisms without incurring high data movement costs**
   - **Leverage processing-in-memory (PIM) to reduce data movement**
   - **PIM mitigates data movement overheads by placing computation units nearby or inside memory**

Polynesia: High-Level Overview

Each island includes (1) a replica of data, (2) an optimized execution engine, and (3) a set of hardware resources



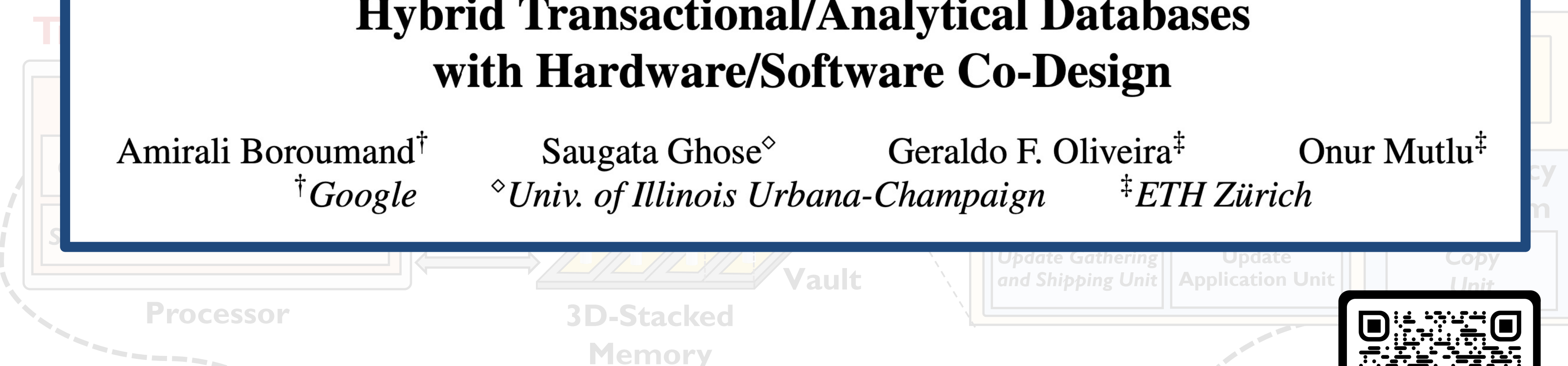
Polynesia: High-Level Overview

Each island includes (1) a replica of data, (2) an optimized execution engine, and (3) a set of hardware resources

Designed to provide high read throughput

Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand[†] Saugata Ghose[◇] Geraldo F. Oliveira[‡] Onur Mutlu[‡]
[†]*Google* [◇]*Univ. of Illinois Urbana-Champaign* [‡]*ETH Zürich*



Full Draft

Conventional multicore CPUs with multi-level caches

Take advantage of PIM to

data movement bott

Key Results

Polynesia achieves 91.6% the transactional throughput of an ideal system by employing custom PIM logic for data freshness/consistency, which significantly reduces resource contention and data movement

Polynesia improves analytical throughput by 63.8% over an optimized multiple-instance system, by eliminating data movement, and using custom logic for update propagation and consistency

Overall, Polynesia achieves all three properties of HTAP system and has a higher transactional/analytical throughput (1.7x/3.74x) over prior HTAP systems

Conclusion

- **<u>Context</u>: Many applications need to perform real-time data analysis using an <u>H</u>ybrid <u>T</u>ransactional/<u>A</u>nalytical <u>P</u>rocessing (HTAP) system**
  - An ideal HTAP system should have **three properties**: (1) **data freshness** and **consistency**, (2) **workload-specific optimization**, (3) **performance isolation**

- **<u>Problem</u>: Prior works cannot achieve all properties of an ideal HTAP system**

- **<u>Key Idea</u>: Divide the system into transactional and analytical processing islands**
  - Enables **workload-specific optimizations** and **performance isolation**

- **<u>Key Mechanism</u>: Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases**
  - Implements **custom algorithms and hardware** to reduce the costs of **data freshness** and **consistency**
  - Exploits **PIM** for analytical processing to alleviate **data movement**

- **<u>Key Results</u>: Polynesia outperforms three state-of-the-art HTAP systems**
  - Average transactional/analytical throughput improvements of **1.7x/3.7x**
  - **48%** reduction on energy consumption

Polynesia:
Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand
Geraldo F. Oliveira
Saugata Ghose
Onur Mutlu

ICDE
2022

Full Draft

SAFARI Google UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN ETH Zürich

Executive Summary

- **Context:** Many applications need to perform real-time data analysis using an Hybrid Transactional/Analytical Processing (HTAP) system
  - An ideal HTAP system should have **three properties**:
    (1) **data freshness** and **consistency**, (2) **workload-specific optimization**,
    (3) **performance isolation**

- **Problem:** HTAP systems **cannot achieve all three** HTAP properties

- **Key Idea:** Divide the system into transactional and analytical **processing islands**
  - Enables **workload-specific optimizations** and **performance isolation**

- **Key Mechanism:** Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases
  - Implements **custom algorithms and hardware** to reduce the costs of **data freshness** and **consistency**
  - Exploits **PIM** for analytical processing to alleviate **data movement**

- **Key Results:** Polynesia outperforms three state-of-the-art HTAP systems
  - Average transactional/analytical throughput improvements of **1.7x/3.7x**
  - **48%** reduction on energy consumption

Outline

Outline

Real-Time Analysis

An explosive interest in many applications domains to perform data analytics on the most recent version of data (real-time analysis)

Use transactions to record each periodic sample of data from all sensors

Run analytics across sensor data to make real-time steering decisions

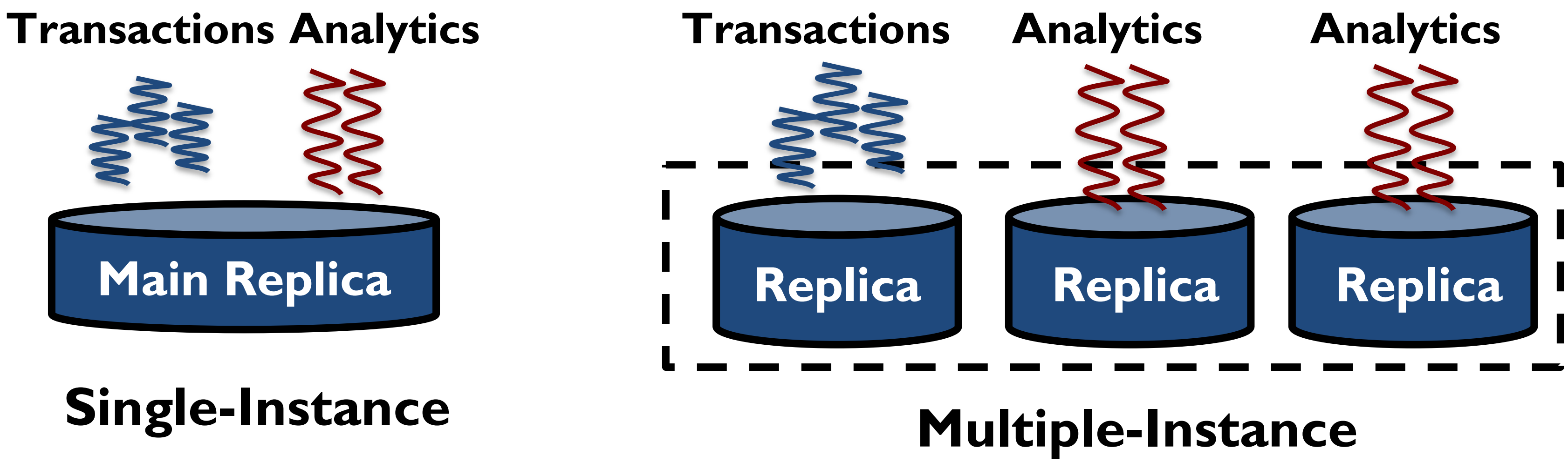
Self-Driving Cars

For these applications, it is critical to analyze the transactions in real-time as the data's value diminishes over time

Outline

State-of-the-Art HTAP Systems

We study two major types of HTAP systems:



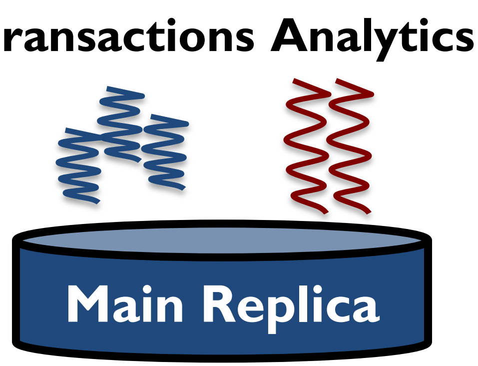
We observe two key problems:

1 Data freshness and consistency mechanisms are costly and cause a drastic reduction in throughput

2 These systems fail to provide performance isolation because of high resource contention

State-of-the-Art HTAP Systems

We study two major types of HTAP systems:



Single-Instance

Multiple-Instance

We observe two key problems:

1 Data freshness and consistency mechanisms are costly and cause a drastic reduction in throughput

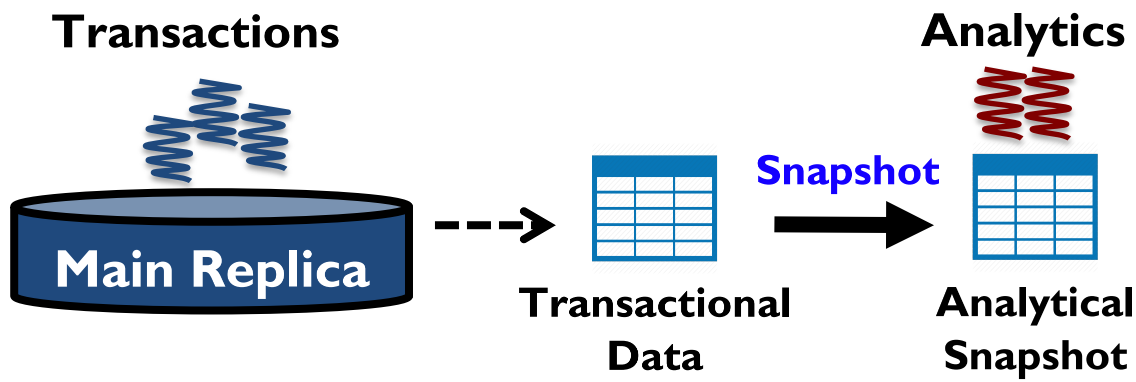
2 These systems fail to provide performance isolation because of high resource contention

Single-Instance: Data Consistency

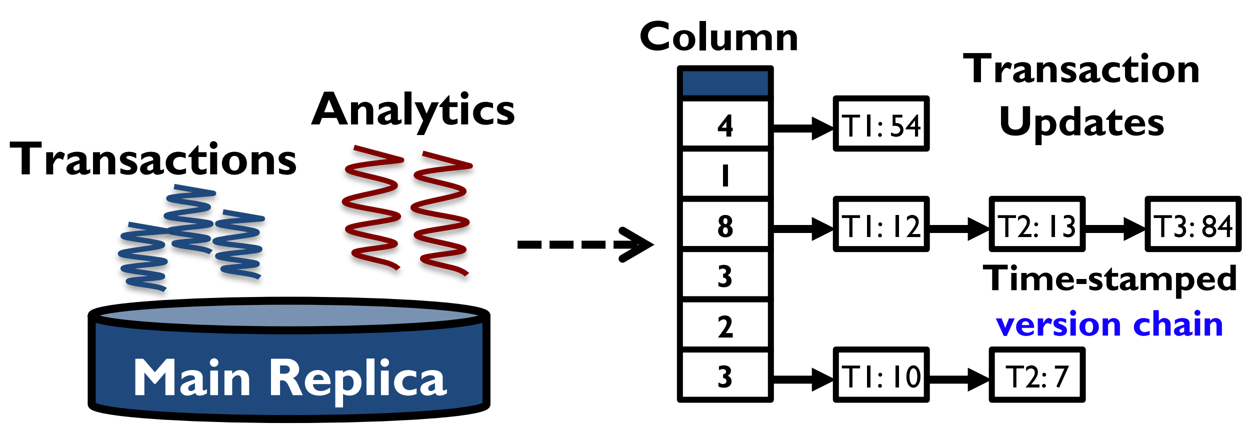
Since both analytics and transactions work on the same data concurrently, we need to ensure that the data is consistent

There are two major mechanisms to ensure consistency:

1 Snapshotting

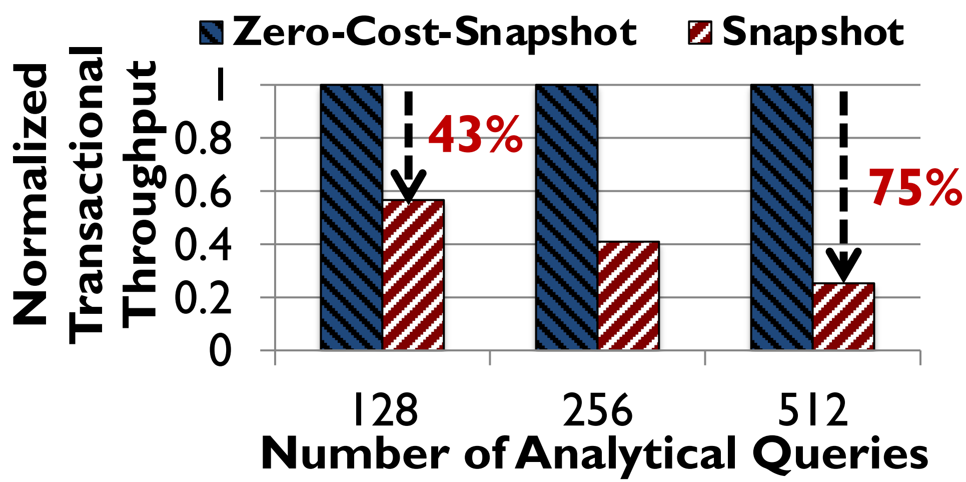


2 Multi-Version Concurrency Control (MVCC)



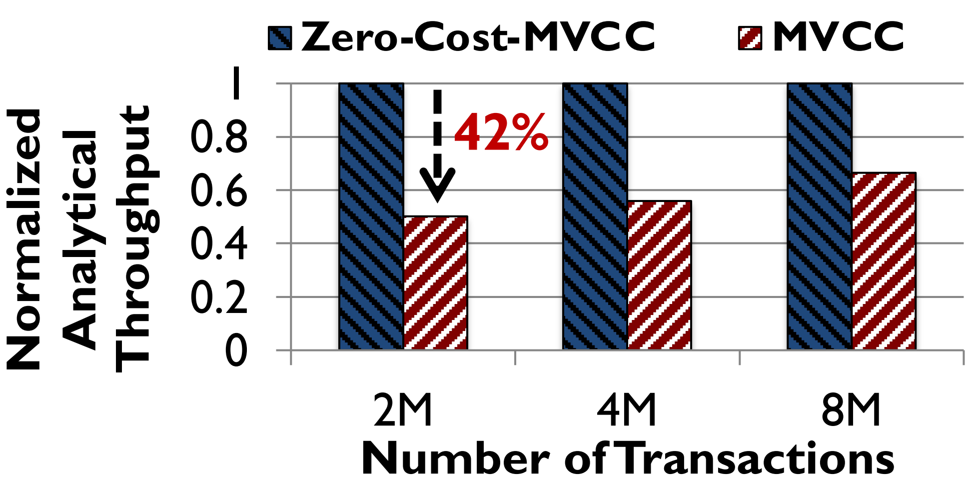
Drawbacks of Snapshotting and MVCC

We evaluate the throughput loss caused by Snapshotting and MVCC:



Throughput loss comes from <u>memcpy</u> operation:

generates a large amount of data movement

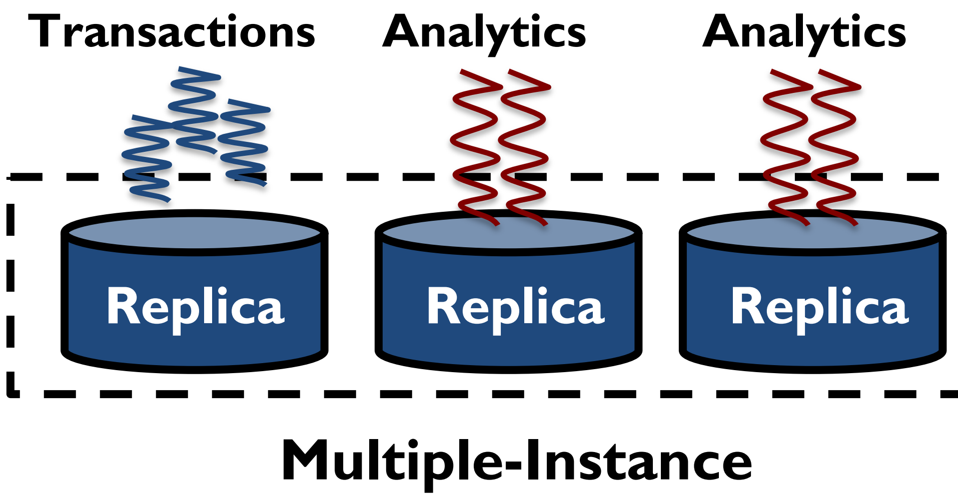


Throughput loss comes from <u>long version chains</u>:

expensive time-stamp comparison and
a large number of random memory accesses

State-of-the-Art HTAP Systems

We study two major types of HTAP systems:



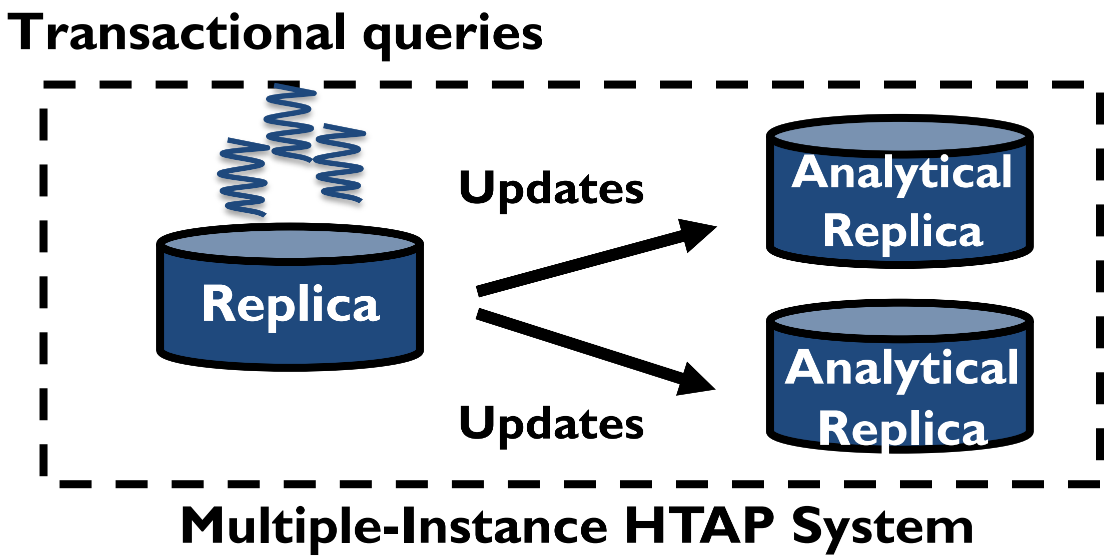
We observe two key problems:

1 Data freshness and consistency mechanisms are costly and cause a drastic reduction in throughput

2 These systems fail to provide performance isolation because of high resource contention

Maintaining Data Freshness

One of the major challenges in multiple-instance systems is to keep analytical replicas up-to-date



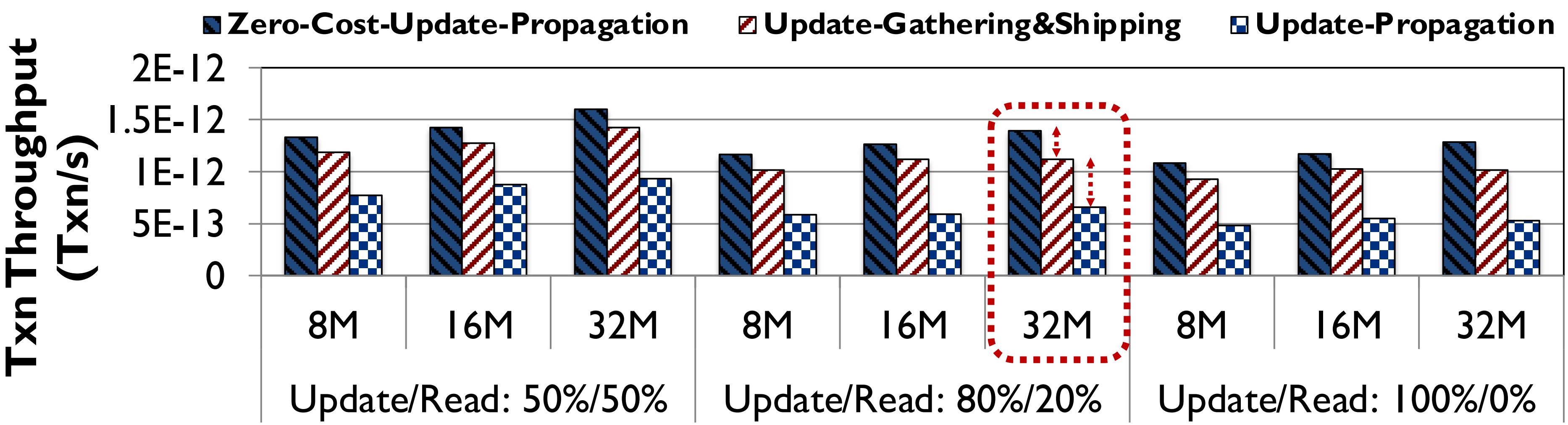
To maintain data freshness (via Update Propagation):

1 **Update Gathering and Shipping**: gather updates from transactional threads and ship them to analytical the replica

2 **Update Application**: perform the necessary format conversation and apply those updates to analytical replicas

Cost of Update Propagation

We evaluate the throughput loss caused by Update Propagation:



Transactional throughput reduces by up to 21.2% during the update gathering & shipping process

Transactional throughput reduces by up to 64.2% during the update application process

Outline

1 Introduction

2 Limitations of HTAP Systems

3 **Polynesia: Overview**

4 Update Propagation

5 Consistency Mechanism

6 Analytical Engine

7 Evaluation

8 Conclusion

Polynesia

Key idea: partition computing resources into two types of isolated and specialized processing islands

↓

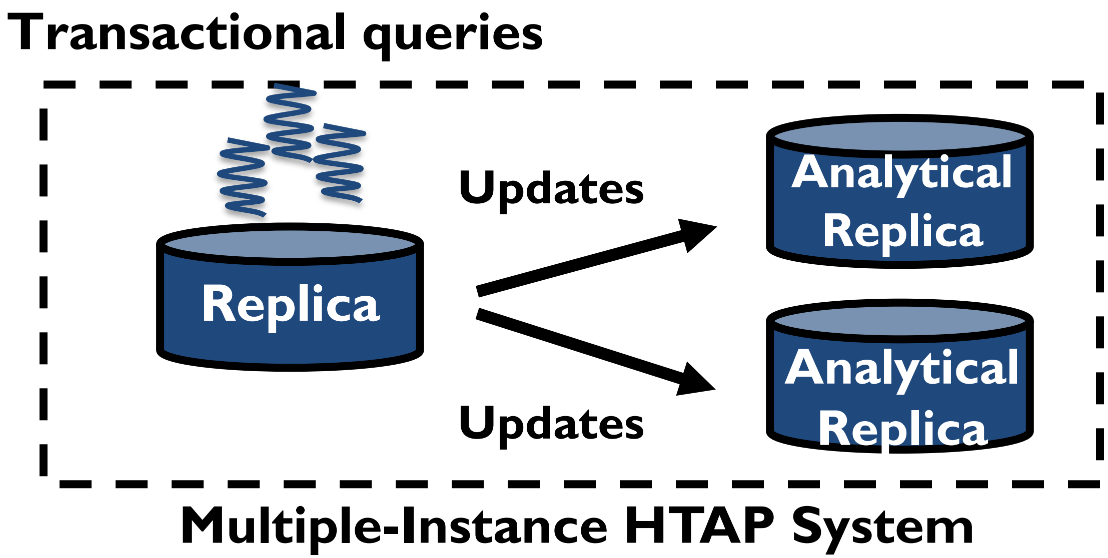
Isolating transactional islands from analytical islands allows us to:

1. **Apply workload-specific optimizations to each island**
2. **Avoid high resource contention**
3. **Design efficient data freshness and consistency mechanisms without incurring high data movement costs**

Outline

Maintaining Data Freshness

One of the major challenges in multiple-instance systems is to keep analytical replicas up-to-date



To maintain data freshness (via Update Propagation):

1 **Update Gathering and Shipping**: gather updates from transactional threads and ship them to analytical the replica

2 **Update Application**: perform the necessary format conversation and apply those updates to analytical replicas

Outline

Outline

Analytical Engine: Query Execution

Efficient analytical query execution strongly depends on:

1. **Data layout and data placement**
2. **Task scheduling policy**
3. **How each physical operator is executed**

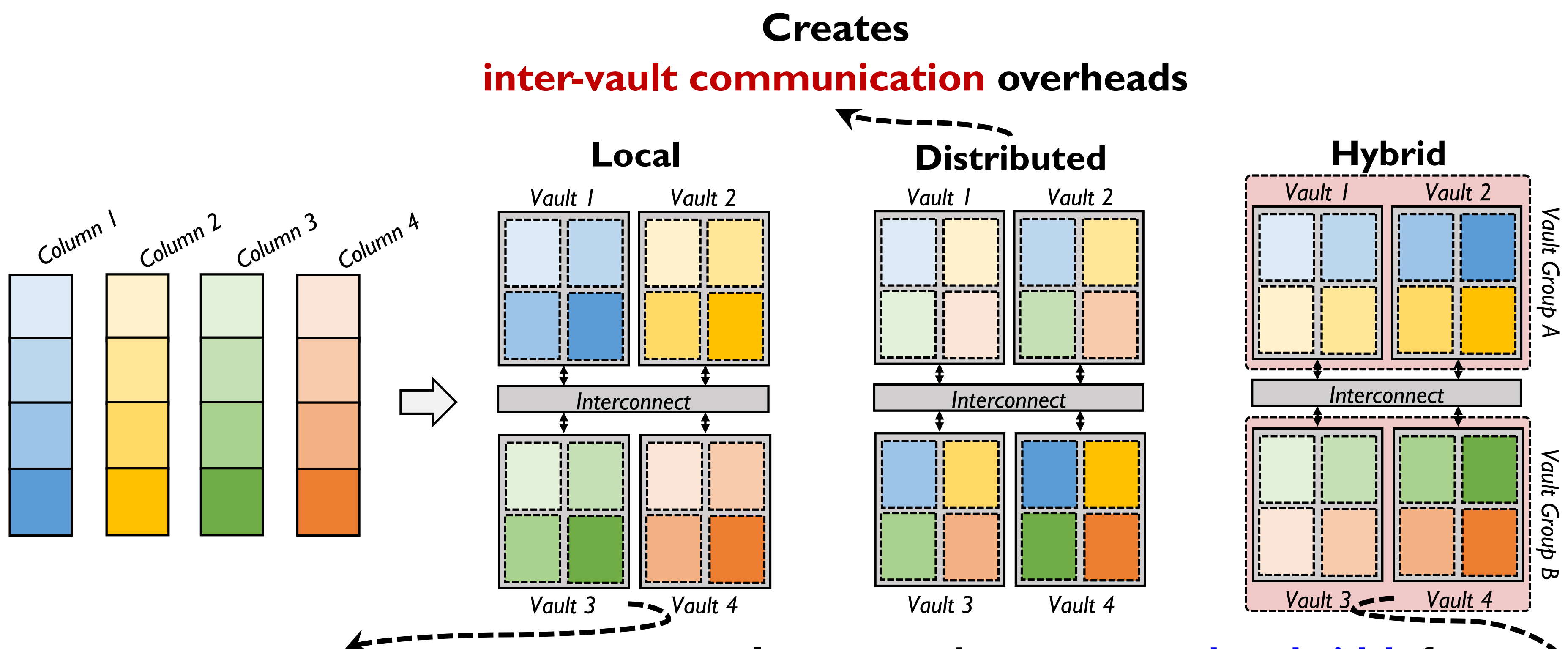
The execution of physical operators of analytical queries significantly benefit from PIM

↓

Without PIM-aware data placement/task scheduler, PIM logic for operators alone cannot provide throughput

Analytical Engine: Data Placement

***Problem*: how to partition analytical data across vaults of the 3D-stacked memory**



Limits the area/power/bandwidth available to the analytical engine inside a vault

Increases the aggregate bandwidth for servicing each query by 4 times, and provides up to 4 times the power/area for PIM logic compared to Local

Analytical Engine: Query Execution

Efficient analytical query execution strongly depends on:

1 Data layout and data placement

2

Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand[†] Saugata Ghose[◇] Geraldo F. Oliveira[‡] Onur Mutlu[‡]
[†]*Google* [◇]*Univ. of Illinois Urbana-Champaign* [‡]*ETH Zürich*

3 How each physical operator is execute

We employ the top-down Volcano (Iterator) execution mo
execute physical operations (e.g., scan, filter, join) while resp
operator's dependencies

Full Draft

Outline

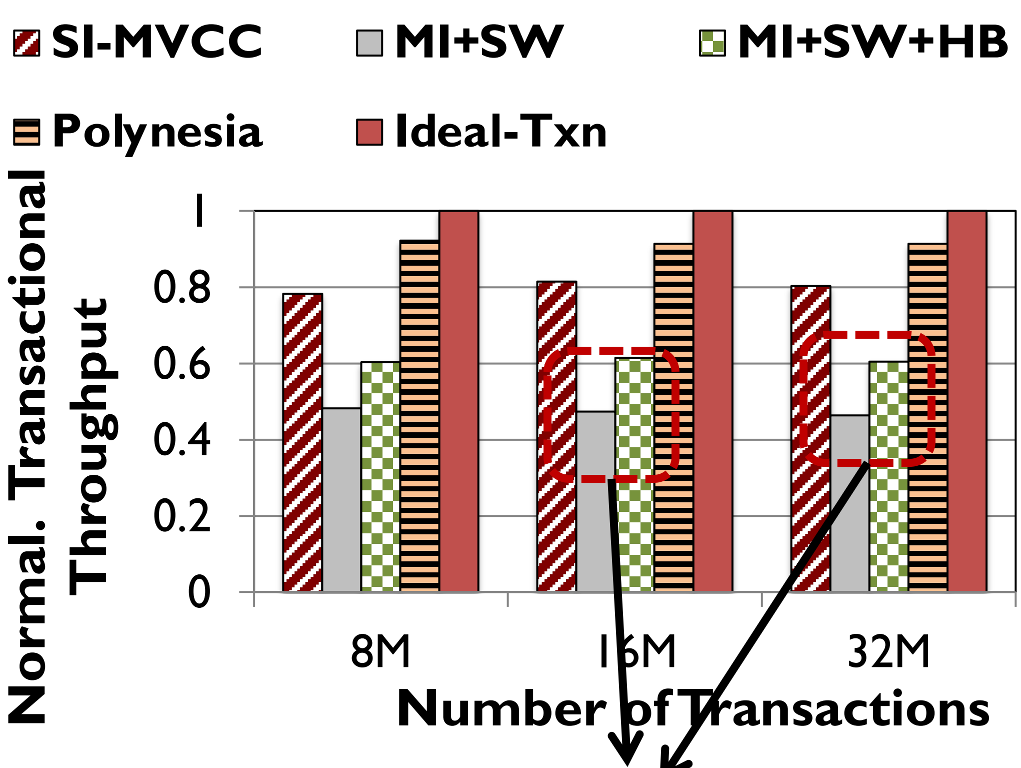
Methodology

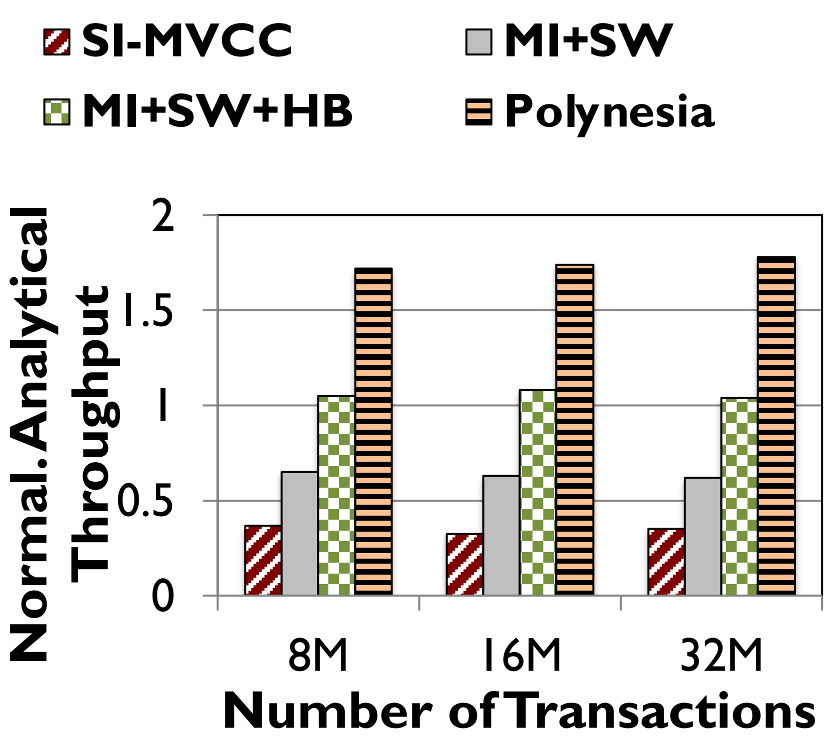
- We adapt previous transactional/analytical engines with our new algorithms
  - **DBx1000** for transactional engine
  - **C-store** for analytical engine

- We use **gem5** to simulate Polynesia
  - Available at: https://github.com/CMU-SAFARI/Polynesia

- We compare Polynesia against:
  - Single-Instance-Snapshot (**SI-SS**): modeled after Hyper
  - Single-Instance-MVCC (**SI-MVCC**): modeled after AnkerDB
  - Multiple-Instance + Polynesia's new algorithms (**MI+SW**)
  - **MI+SW+HB: MI+SW** with a 256 GB/s main memory device
  - **Ideal-Txn**: the peak transactional throughput if transactional workloads run in isolation

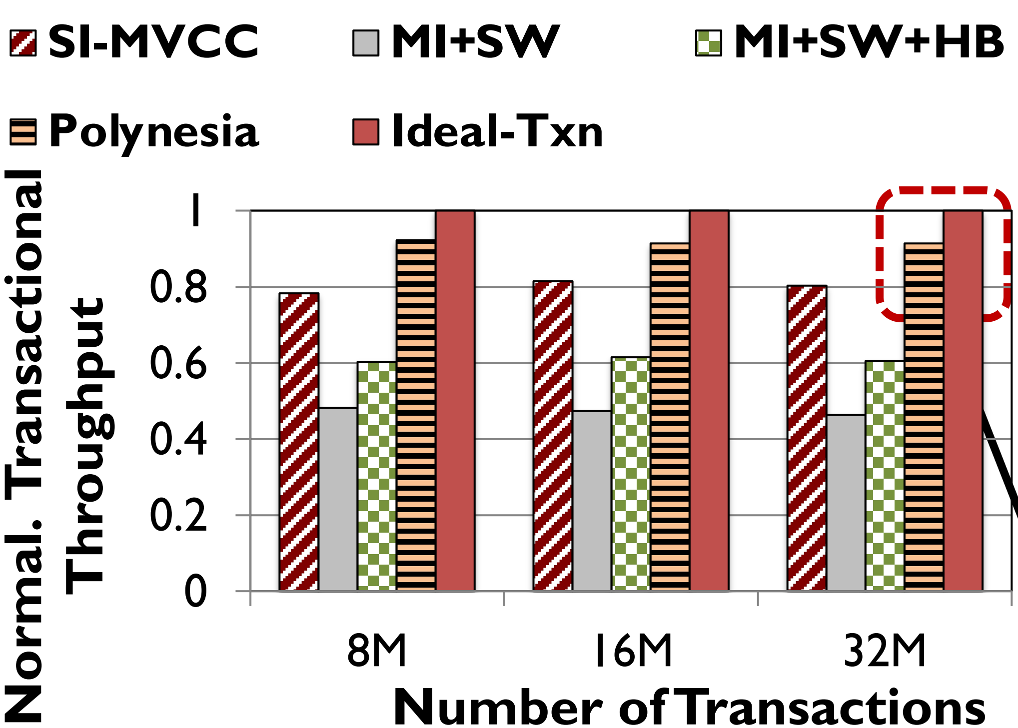
End-to-End System Analysis (2/6)

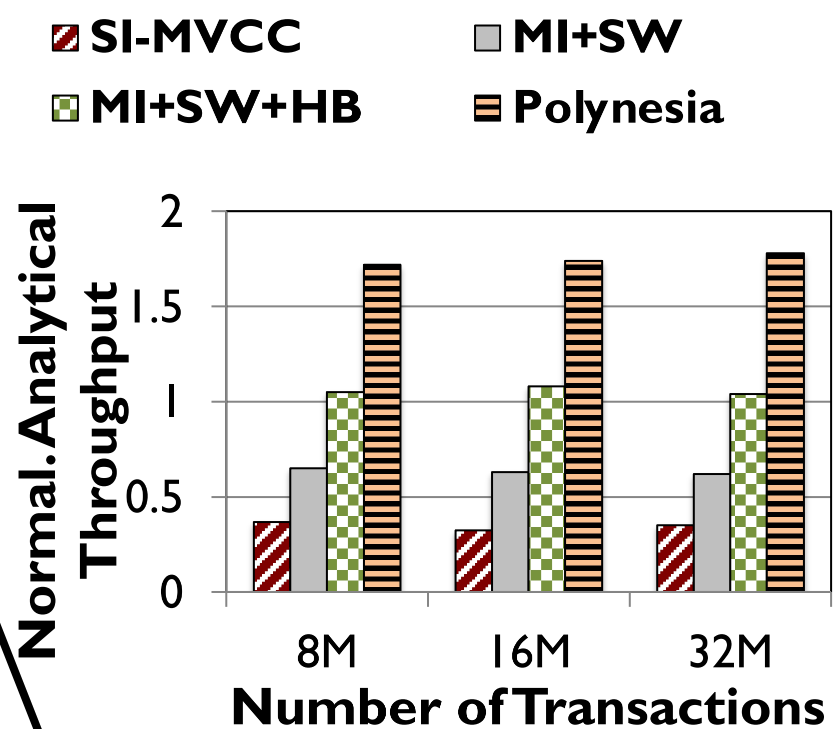




Both MI+SW and MI+SW+HB fall **significantly short** of Ideal-Txn because of lack of **performance isolation** and overhead of **update propagation**

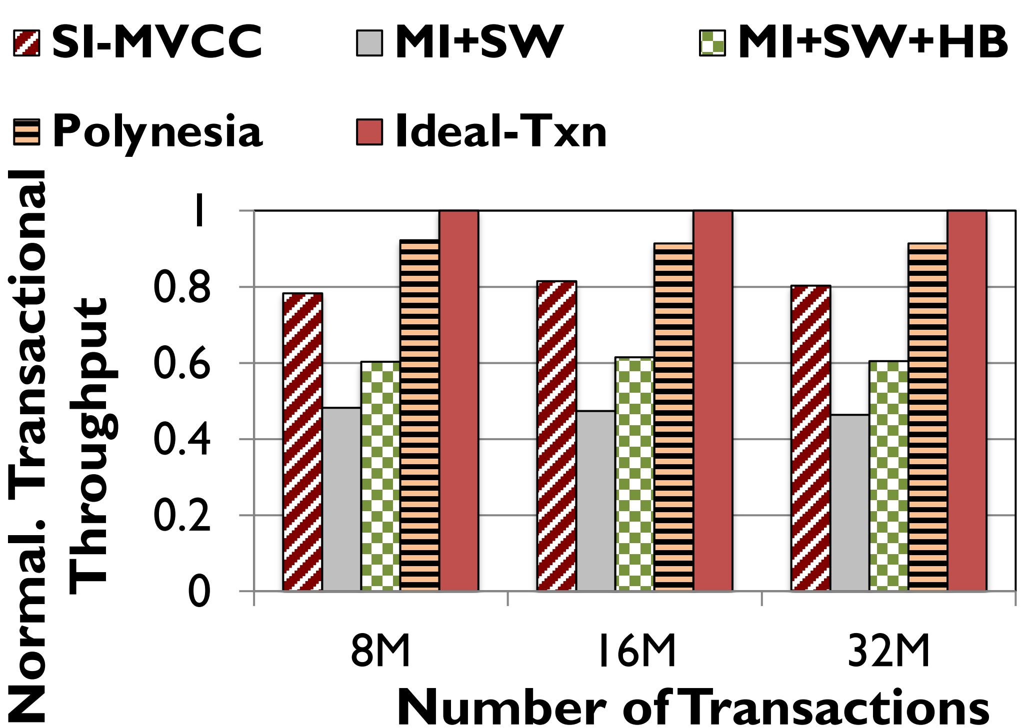
End-to-End System Analysis (3/6)

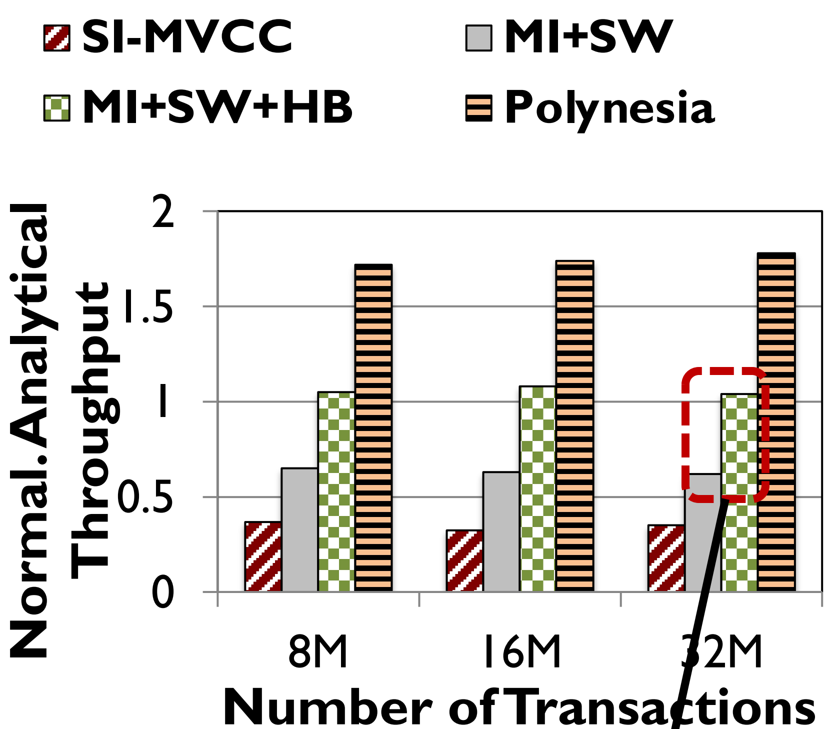




Polynesia comes within 8.4% of ideal Txn because it uses custom PIM logic for data freshness/consistency mechanisms which significantly reduce resource contention and data movement

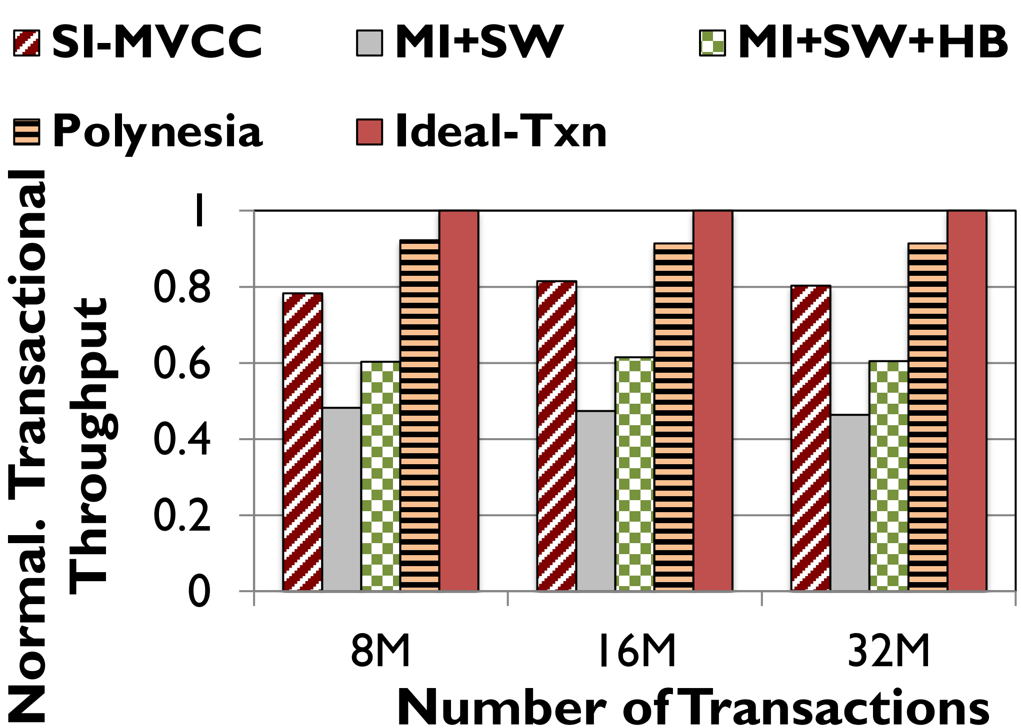
End-to-End System Analysis (4/6)

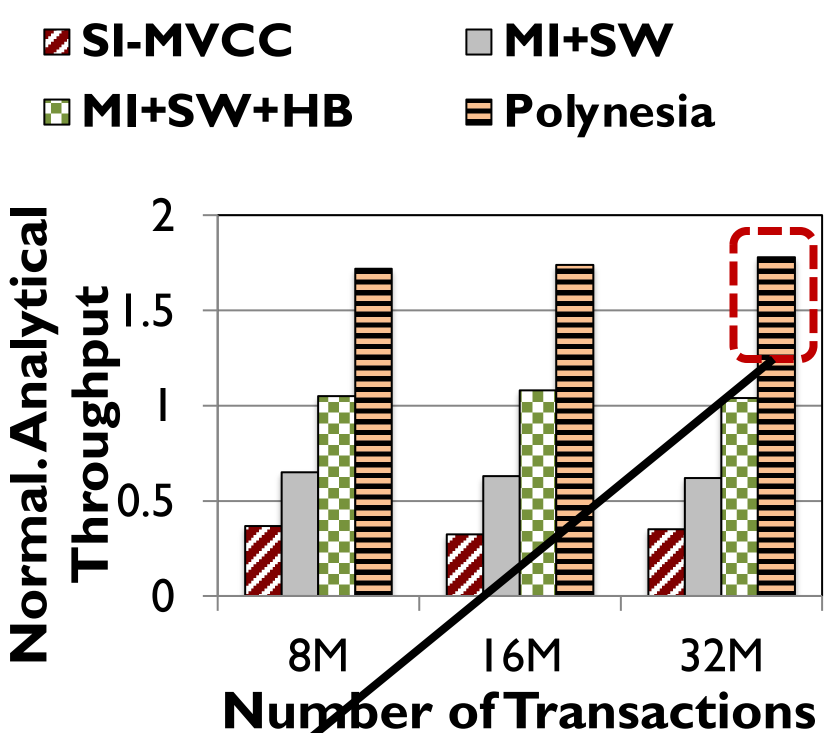




MI+SW+HB is the best software-only HTAP for analytical workloads, because it provides workload-specific optimizations, but it still loses 35.3% of the analytical throughput due to high resource contention

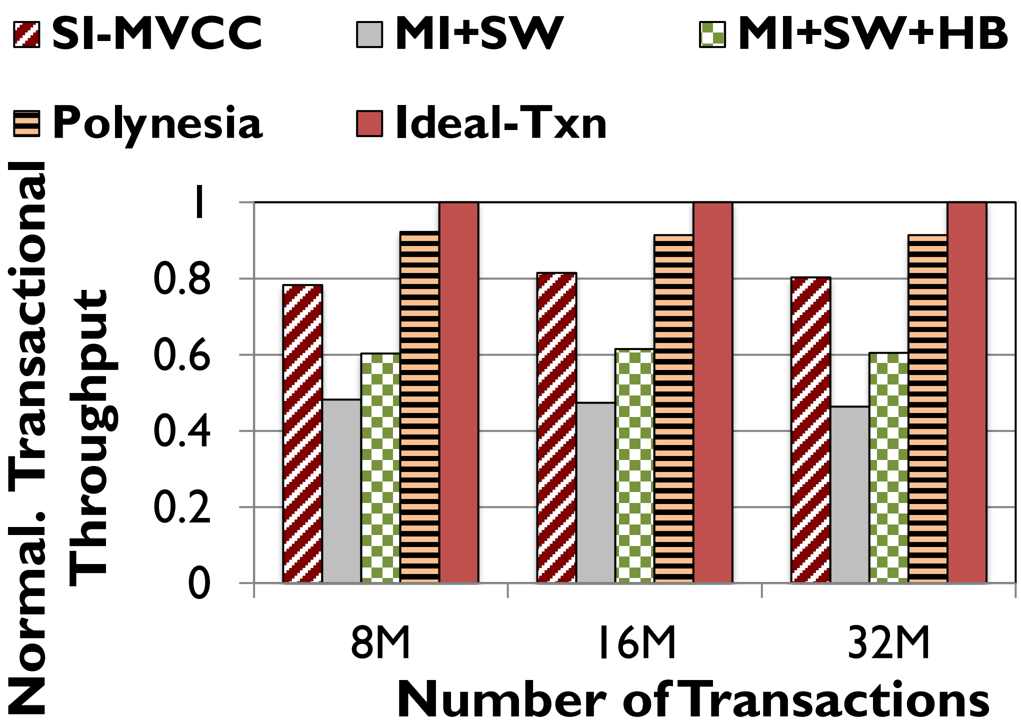
End-to-End System Analysis (5/6)

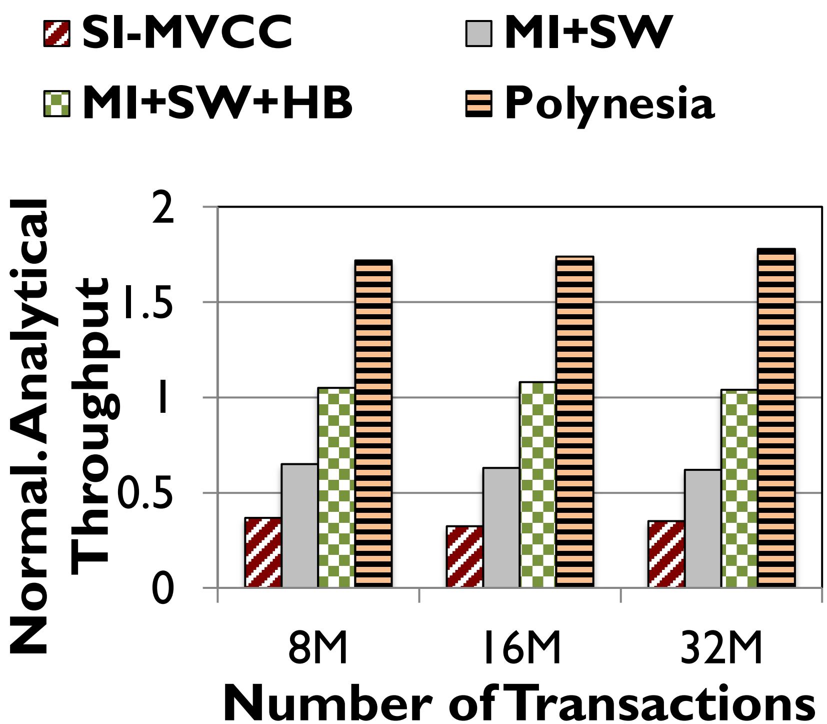




Polynesia improves over MI+SW+HB by 63.8%, by eliminating data movement, and using custom logic for update propagation and consistency

End-to-End System Analysis (6/6)





Overall, Polynesia achieves all three properties of HTAP system and has a higher transactional/analytical throughput (1.7x/3.74x) over prior HTAP systems

More in the Paper

- Real workload analysis
- Effect of the update propagation technique
- Effect of the consistency mechanism
- Effect of the analytical engine
- Effect of the dataset size
- Energy analysis
- Area analysis

More in the Paper

- Real workload analysis
- Effect of the update propagation technique
- Effect of the dataset size
- Energy analysis
- Area analysis

Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand[†] Saugata Ghose[◇] Geraldo F. Oliveira[‡] Onur Mutlu[‡]
[†]*Google* [◇]*Univ. of Illinois Urbana-Champaign* [‡]*ETH Zürich*

Full Draft

Outline

Conclusion

- **Context**: Many applications need to perform real-time data analysis using an Hybrid Transactional/Analytical Processing (HTAP) system
  - An ideal HTAP system should have **three properties**: (1) **data freshness** and **consistency**, (2) **workload-specific optimization**, (3) **performance isolation**

- **Problem**: HTAP systems **cannot achieve all three** HTAP properties

- **Key Idea**: Divide the system into transactional and analytical **processing islands**
  - Enables **workload-specific optimizations** and **performance isolation**

- **Key Mechanism**: Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases
  - Implements **custom algorithms and hardware** to reduce the costs of **data freshness** and **consistency**
  - Exploits **PIM** for analytical processing to alleviate **data movement**

- **Key Results**: Polynesia outperforms three state-of-the-art HTAP systems:
  - Average transactional/analytical throughput improvements of **1.7x/3.7x**
  - **48%** reduction on energy consumption

Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand
Geraldo F. Oliveira
Saugata Ghose
Onur Mutlu

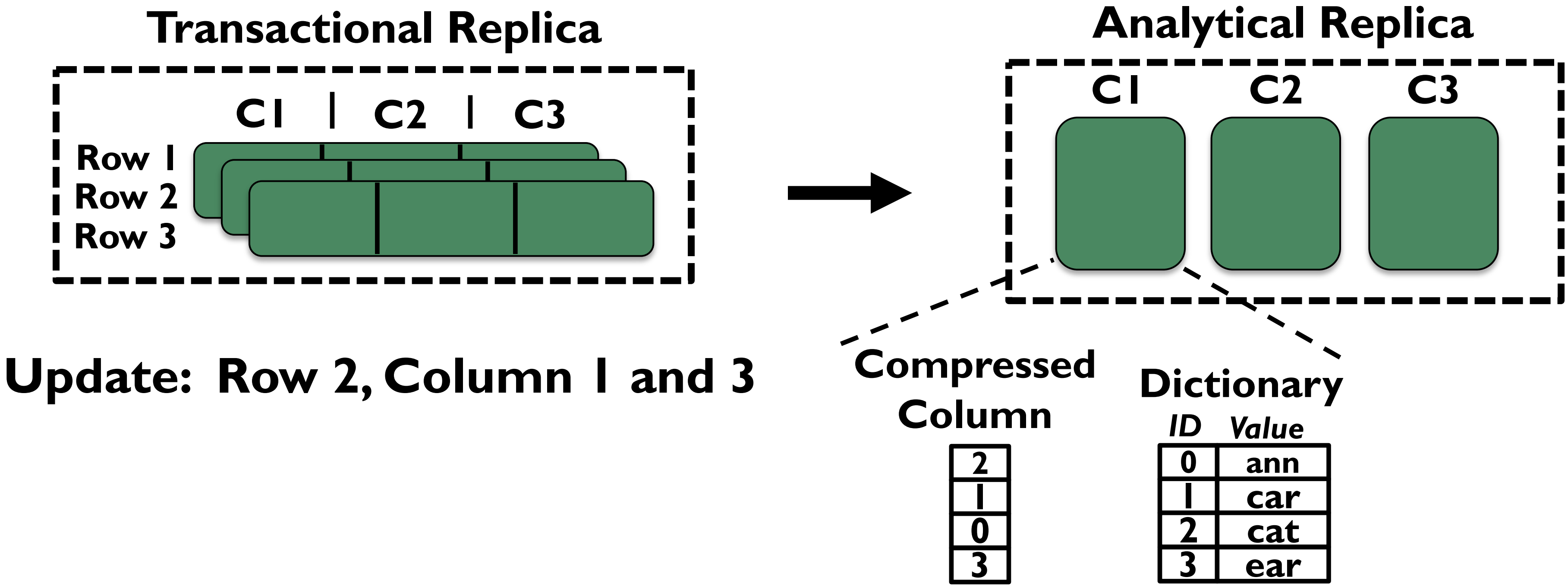
ICDE
2022

Full Draft

SAFARI Google UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN ETH Zürich

Update Propagation: Update Application

***Goal*: perform the necessary format conversation and apply transactional updates to analytical replicas**



1. A simple tuple update in row-wise layout leads to multiple random accesses in column-wise layout
2. Updates change encoded value in the dictionary → (1) Need to reconstruct the dictionary, and (2) recompress the column

Limitations of State-of-the-Art

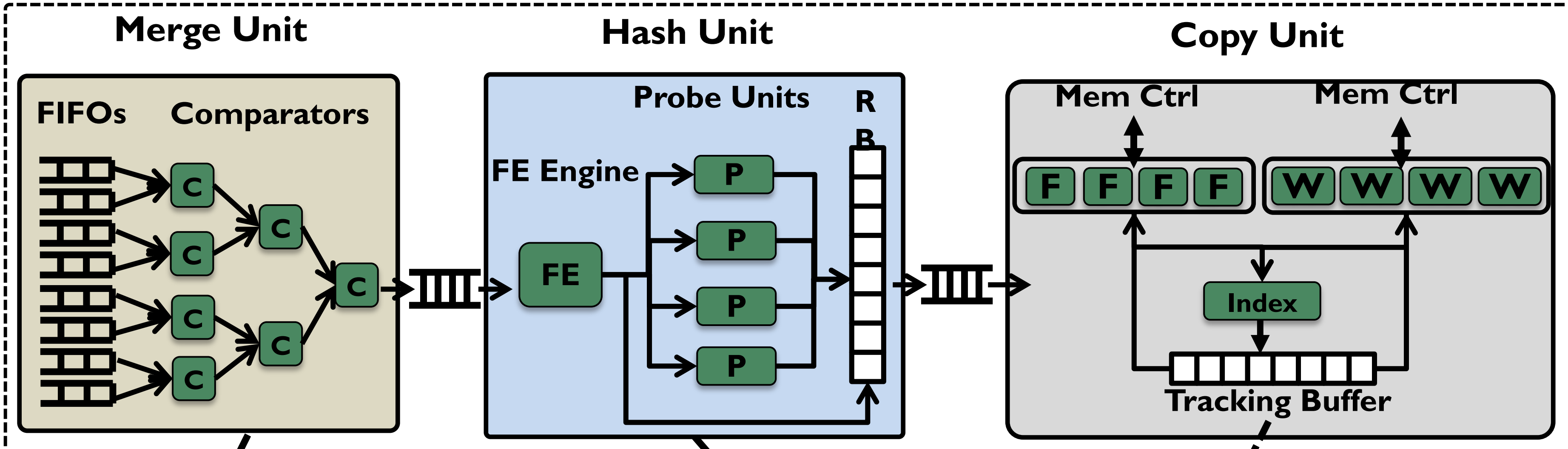
We extensively study state-of-the-art HTAP systems and observe two key problems:

1 **Data freshness and consistency mechanisms generate a large amount of data movement which causes a drastic reduction in transactional/analytical throughput**

2 **They fail to provide performance isolation because of the high resource contention between transactional and analytical workloads**

Update Gathering & Shipping: Hardware

To avoid these bottlenecks, we design a new hardware accelerator, called update shipping unit



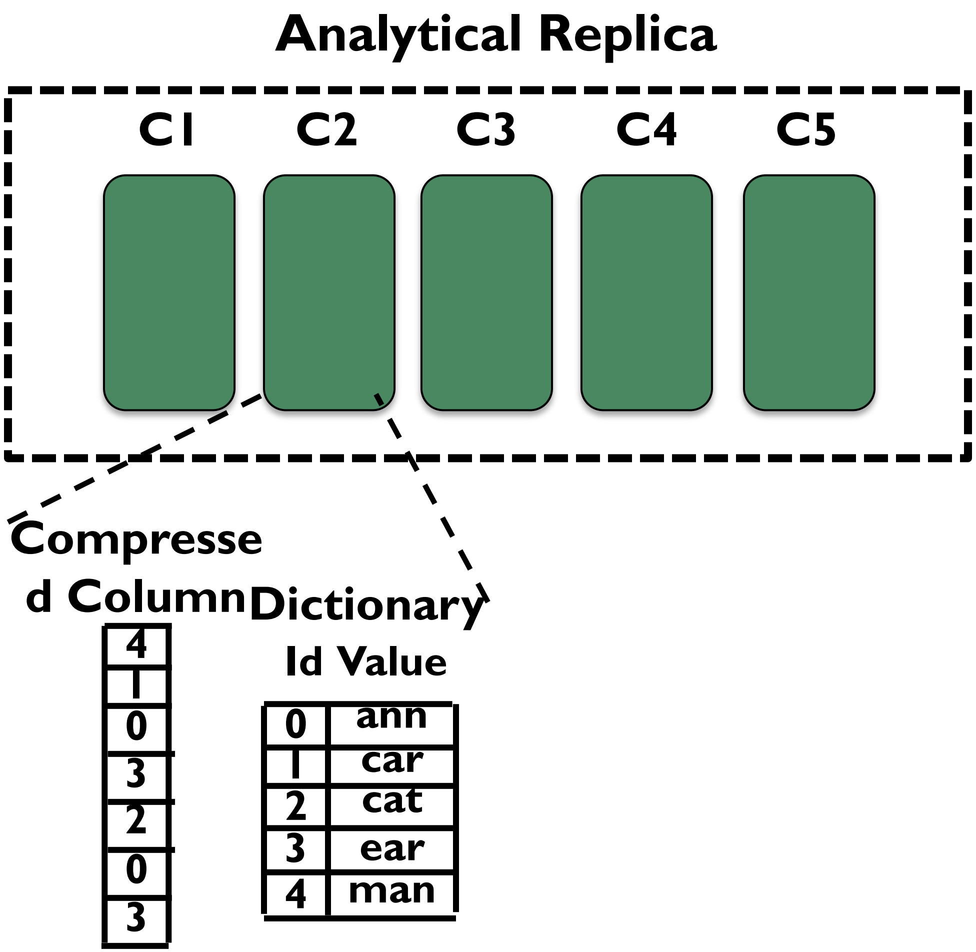
A **3-level comparator tree** to merge updates

Decoupled **hash computation** from the **bucket traversal** to allow for **concurrent** lookups

Multiple **fetch** and **write-back** units to issue multiple memory accesses **concurrently**

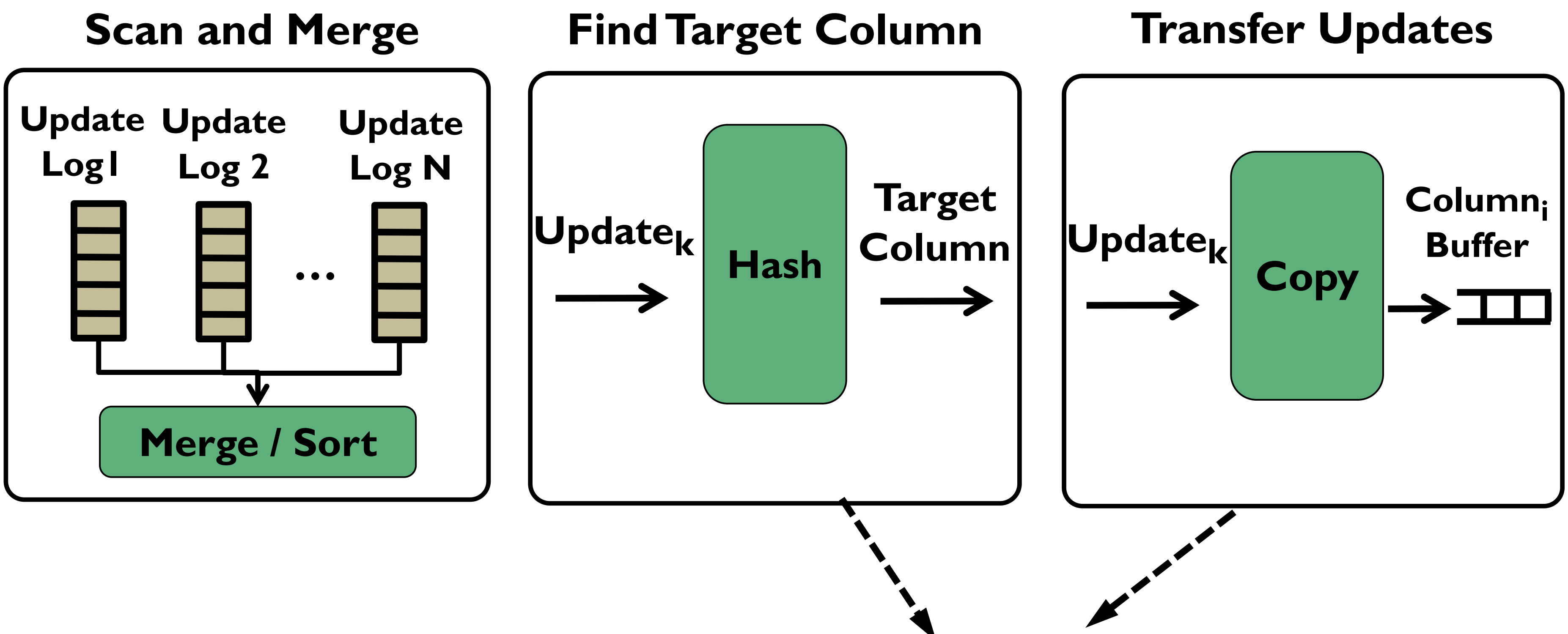
Update Application

Like other relational analytical DBMSs, our analytical engine uses the column-wise data layout and dictionary encoding



Update Gathering & Shipping: Algorithm

Our update shipping algorithm has three major stages:



Two major bottlenecks that keep us from meeting **data freshness** and **performance isolation**

These primitives generate a large amount of data movement and account for 87.2% of our algorithm's execution time

Single-Instance: High Cost of Consistency

Since both analytics and transactions work on the same data concurrently, we need to ensure that the data is consistent

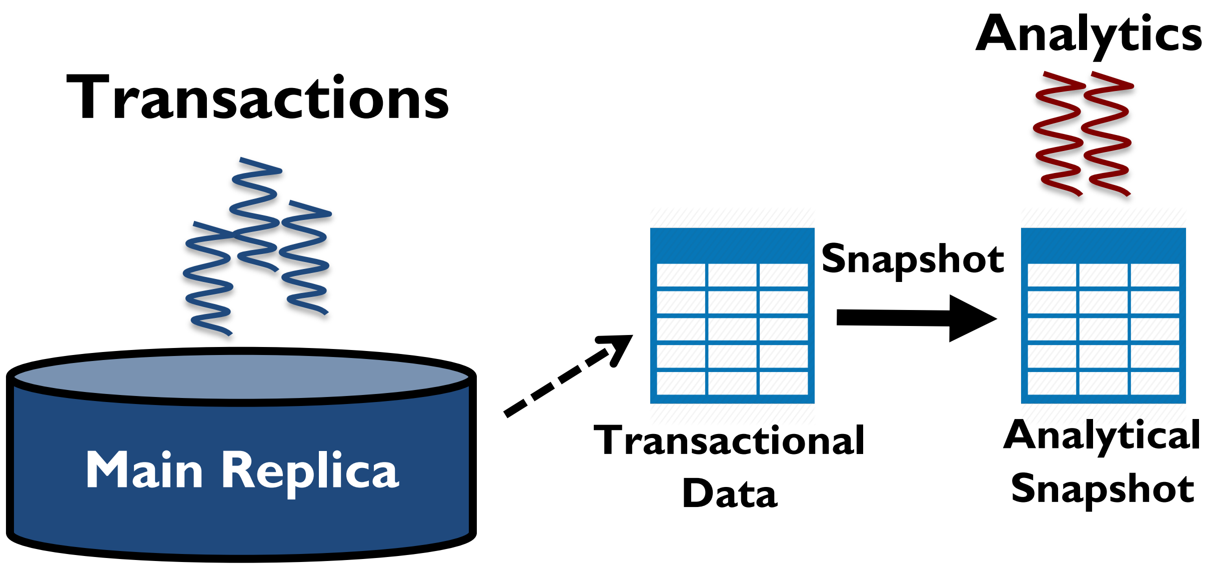
There are two major mechanisms to ensure consistency:

1 **Snapshotting (Snapshot Isolation)**

2 **Multi-Version Concurrency Control (MVCC)**

Snapshotting

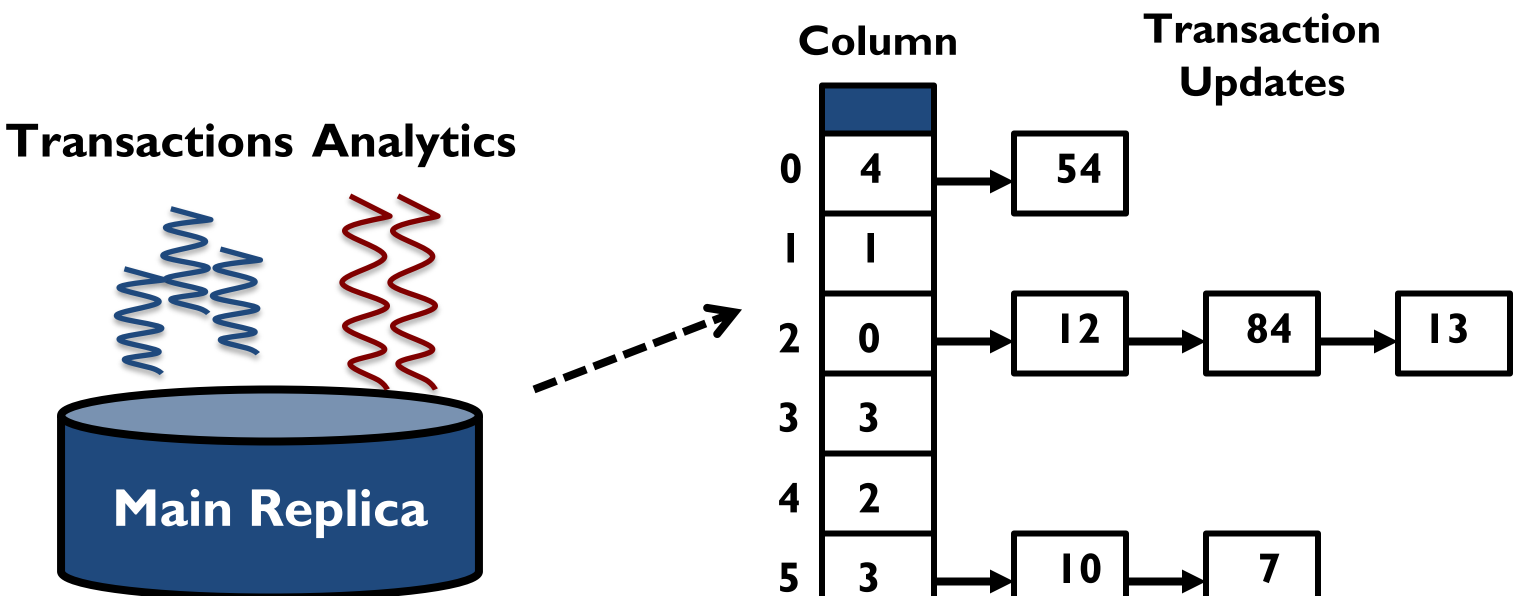
Several HTAP systems use snapshotting to provide consistency via Snapshot Isolation (SI)



These systems explicitly create snapshots from the most recent version of data and let the analytics run on the snapshot while transactions continue updating data

Multi-Version Concurrency Control (MVCC)

MVCC <u>avoids</u> making full copies of data by keeping several versions of the

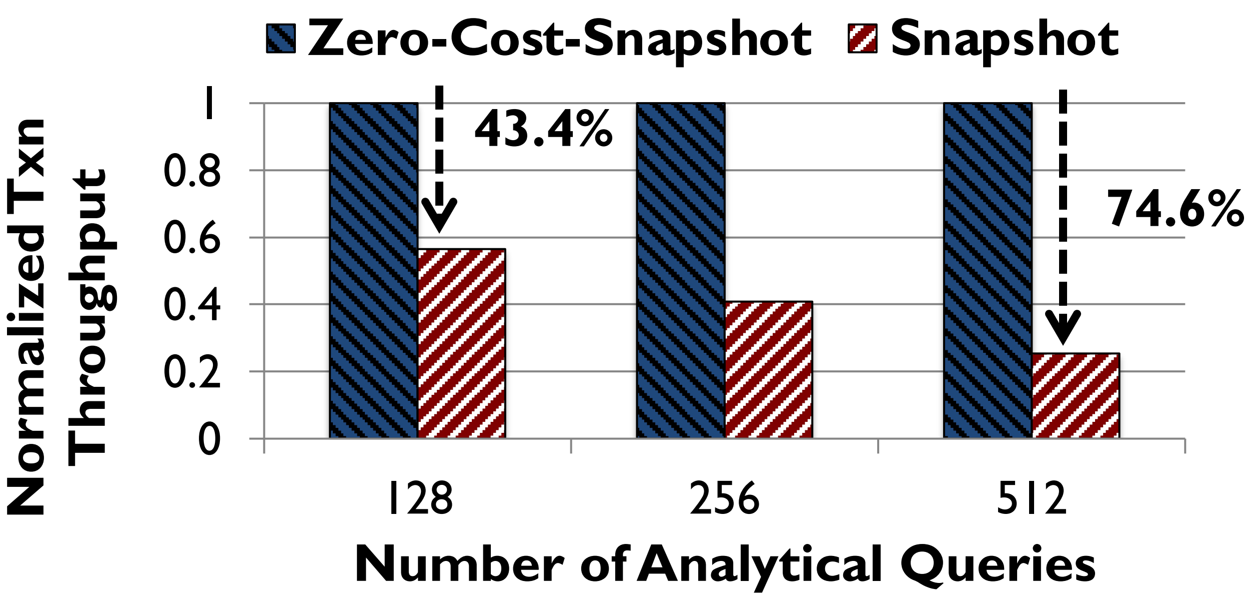


When updates happen, MVCC creates a new time-stamped version of data and keeps <u>the old version</u> in a version chain <u>alongside the data</u>

Snapshotting: Drawbacks

We find that this approach requires frequent snapshot creation to sustain data freshness under high transactional update rate

Two Txn threads
Each 1M Txn queries
Write/read 50%



The overhead comes from Memcpy operation which generates a large amount of data movement and introduces significant interference

More Insights on Data Freshness Challenges

Our analysis shows that simply providing higher bandwidth (8x) to CPU cores does not address the challenges

↓

We need to take advantage of PIM logic to reduce data movement and resource contention

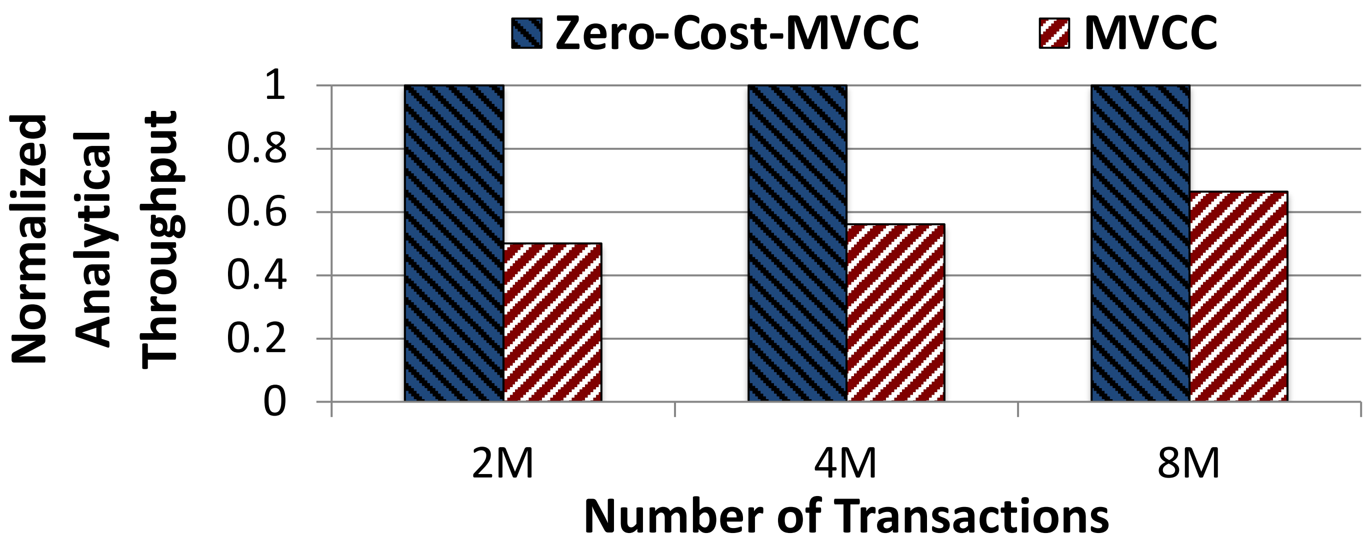
We find that simply offloading them to general purpose PIM cores does not address the challenges

↓

We need to design custom algorithm and hardware to efficiently execute update shipping/application process

Multi-Version Concurrency Control (MVCC)

We observe that MVCC overhead leads to 42.4% performance loss over zero-cost MVCC

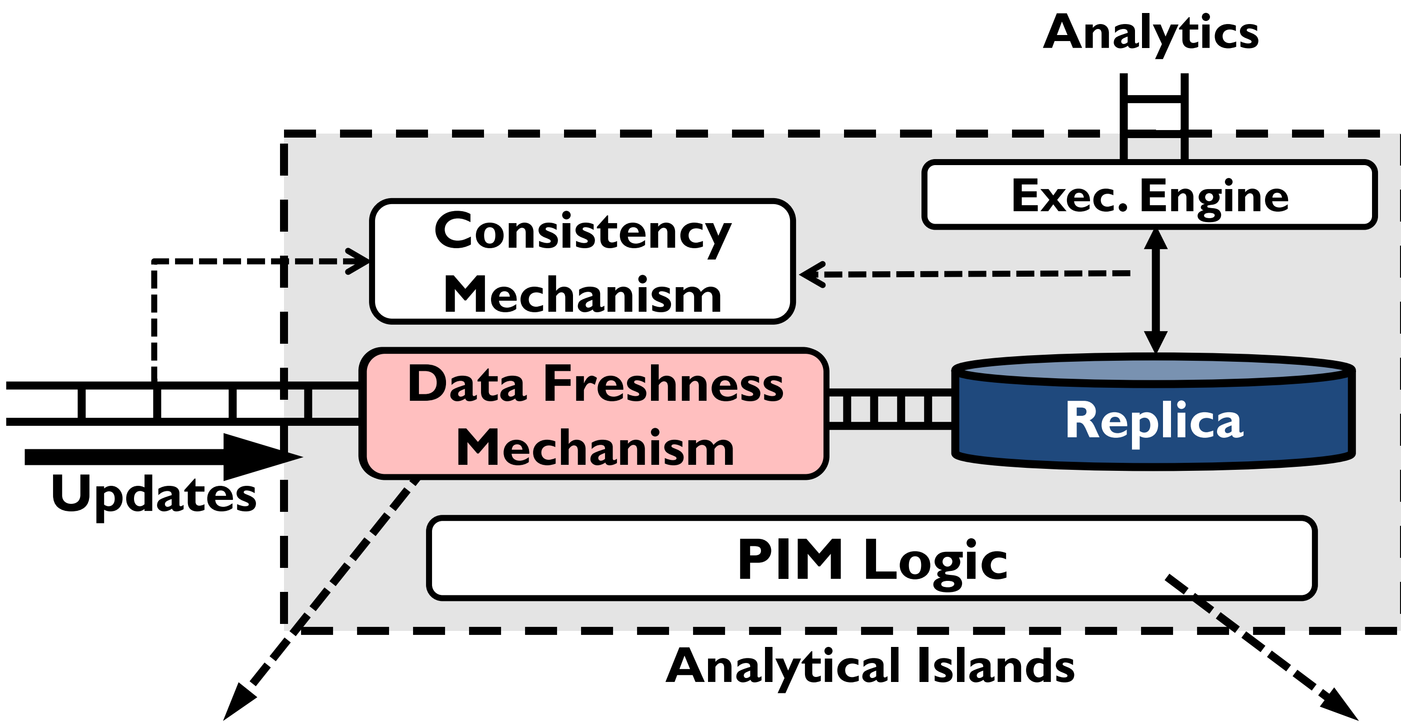


We find that long version chains are the root cause of the issue

1. **Frequent transactional updates create lengthy version chains**
2. **Scan-heavy analytics traverses a lengthy version chain upon accessing a data tuple**
   - **Expensive time-stamp comparison + a very large number of random memory accesses**

Analytical Islands Key Components

We co-design new algorithms and efficient hardware support for the three key components of an analytical island



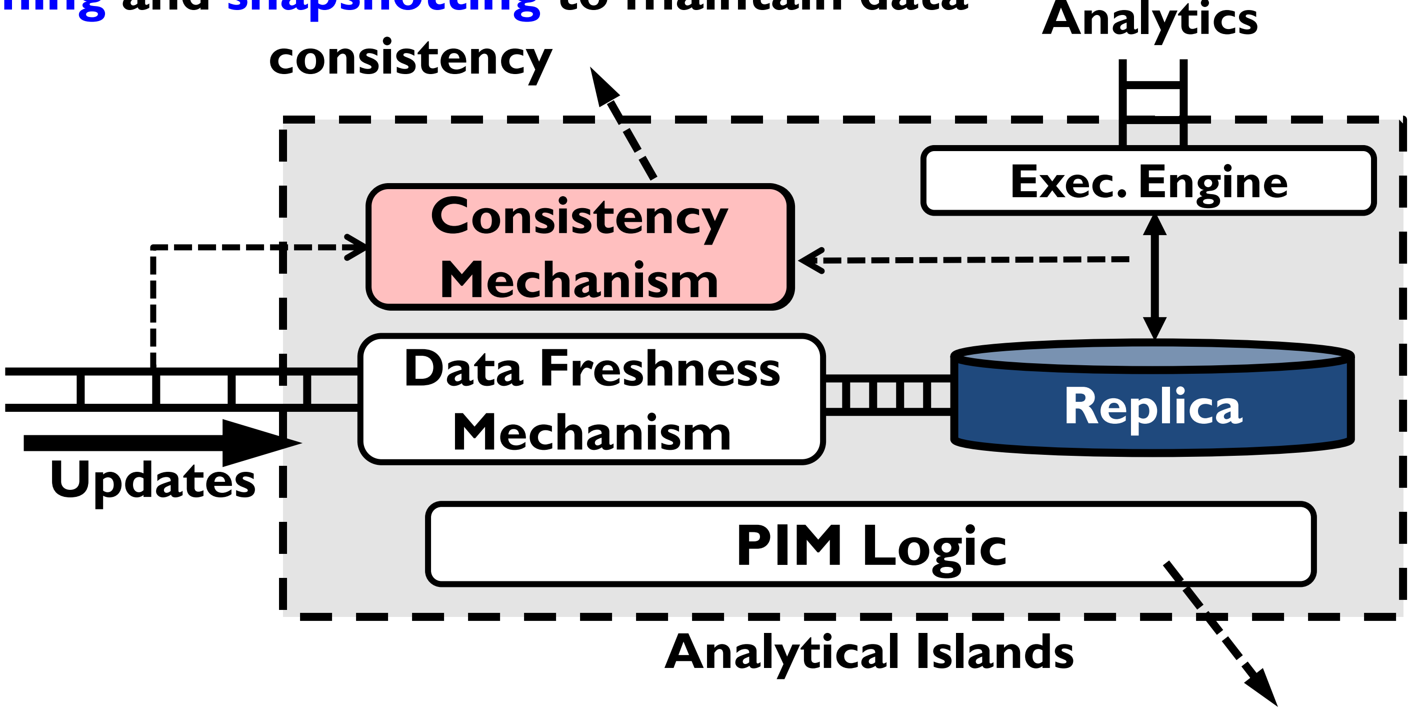
Design two algorithms:
(1) update shipping and (2) update application

Design custom PIM logic for both algorithms

Analytical Islands Key Components

We co-design new algorithms and efficient hardware support for the three key components of an analytical island

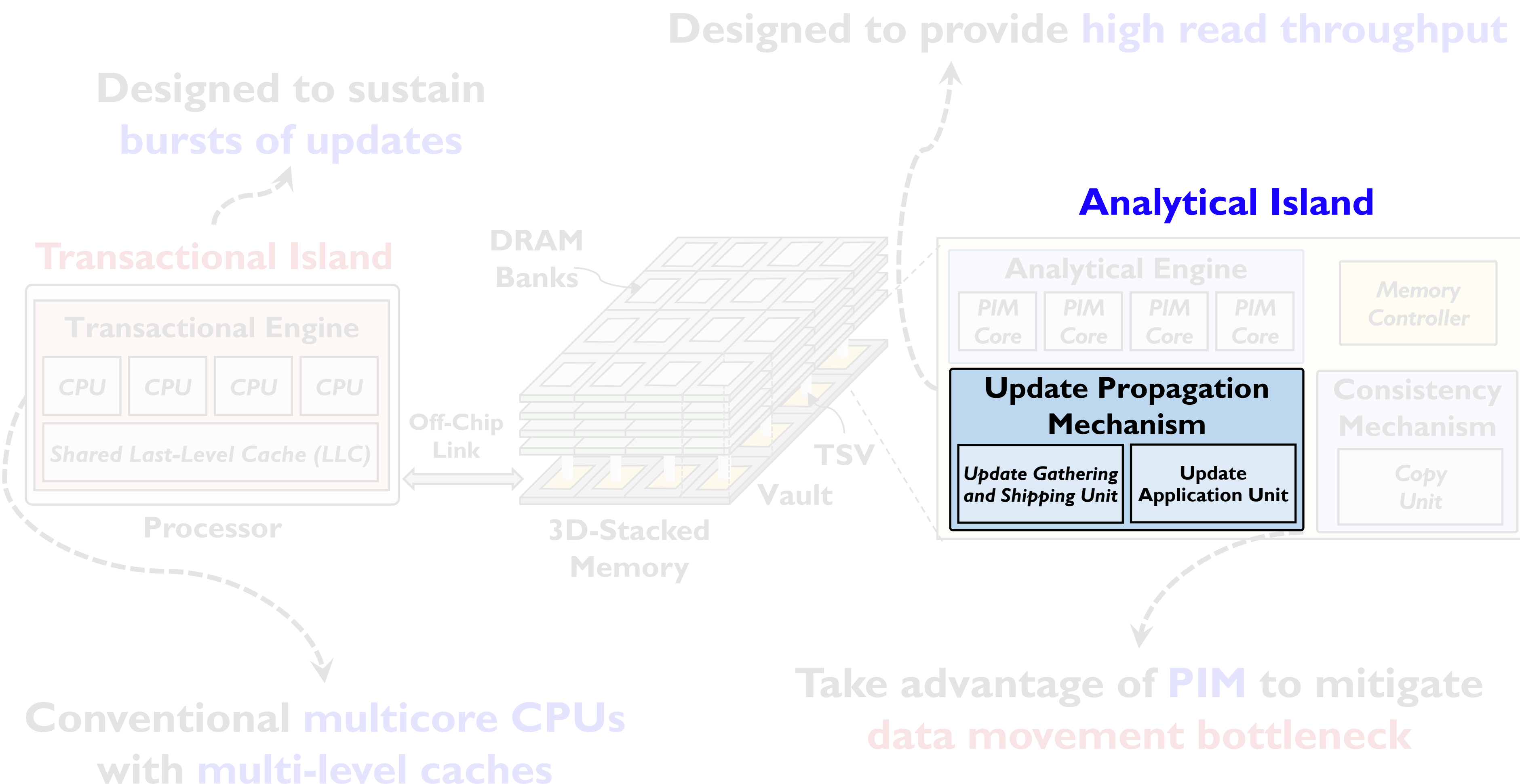
Develop an algorithm relies on a combination of versioning and snapshotting to maintain data consistency



Design an in-memory copy unit that enables highly efficient snapshot creation

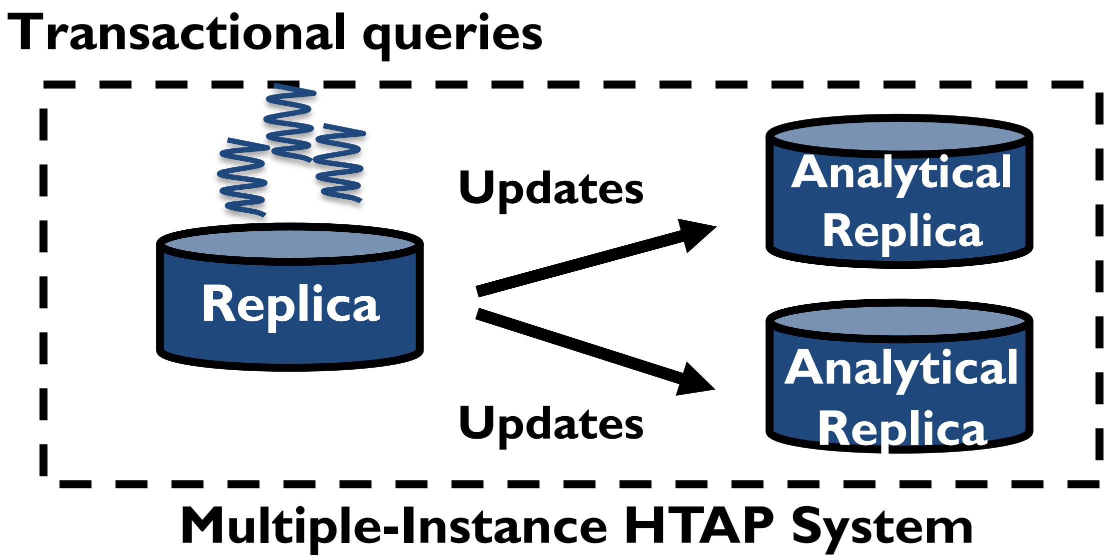
Polynesia: High-Level Overview

Each island includes (1) a replica of data, (2) an optimized execution engine, and (3) a set of hardware resources



Maintaining Data Freshness

One of the major challenges in multiple-instance systems is to keep analytical replicas up-to-date



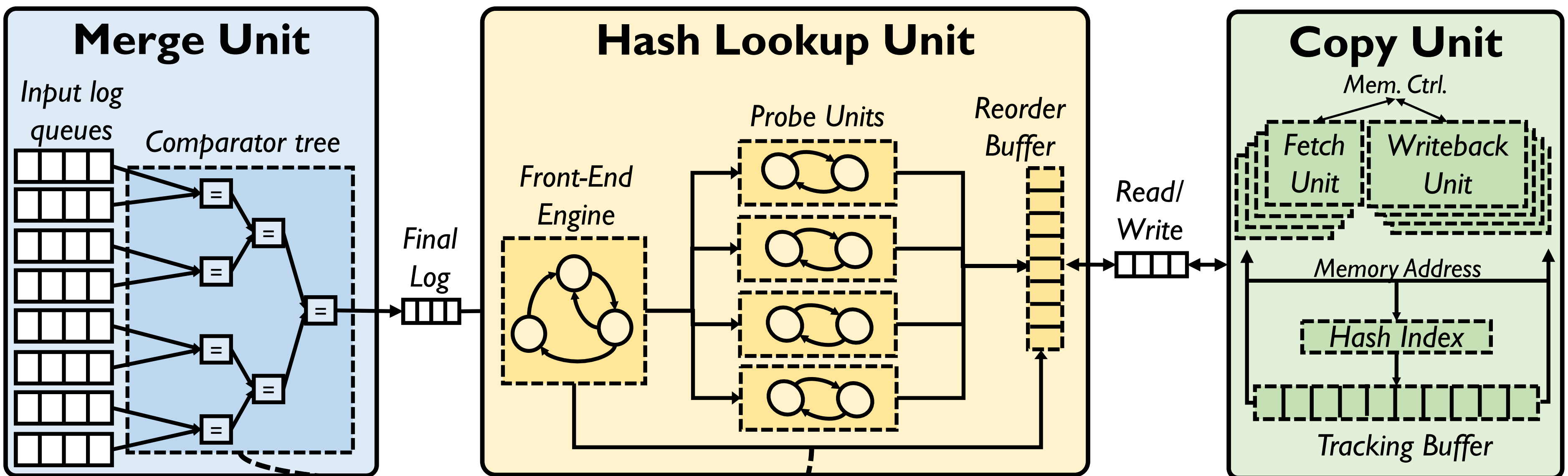
To maintain data freshness (via Update Propagation):

1 **Update Gathering and Shipping: gather updates from transactional threads and ship them to analytical the replica**

2 **Update Application: perform the necessary format conversation and apply those updates to analytical replicas**

Update Gathering & Shipping

We co-design a new software/hardware accelerator, called update gathering & shipping unit



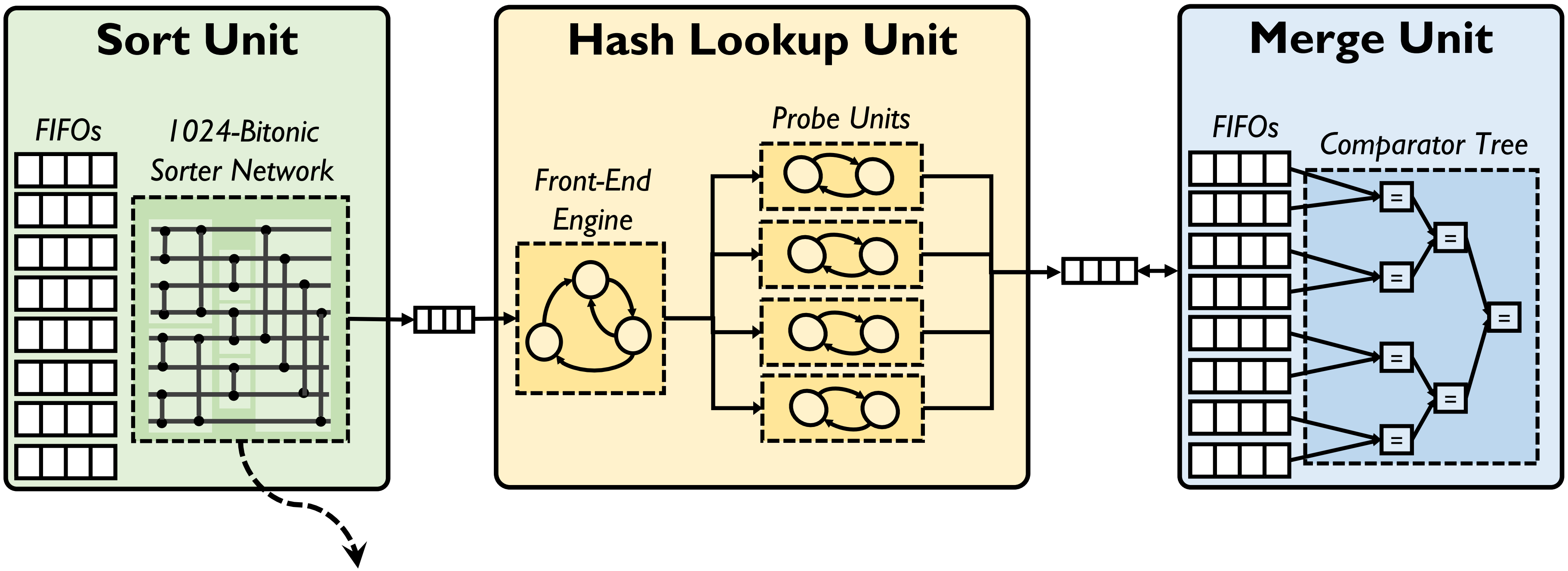
A 3-level comparator tree to merge updates

Decoupled hash computation from the hash bucket traversal to allow for concurrent hash lookups

Multiple fetch and write-back units to issue multiple memory accesses concurrently

Update Application

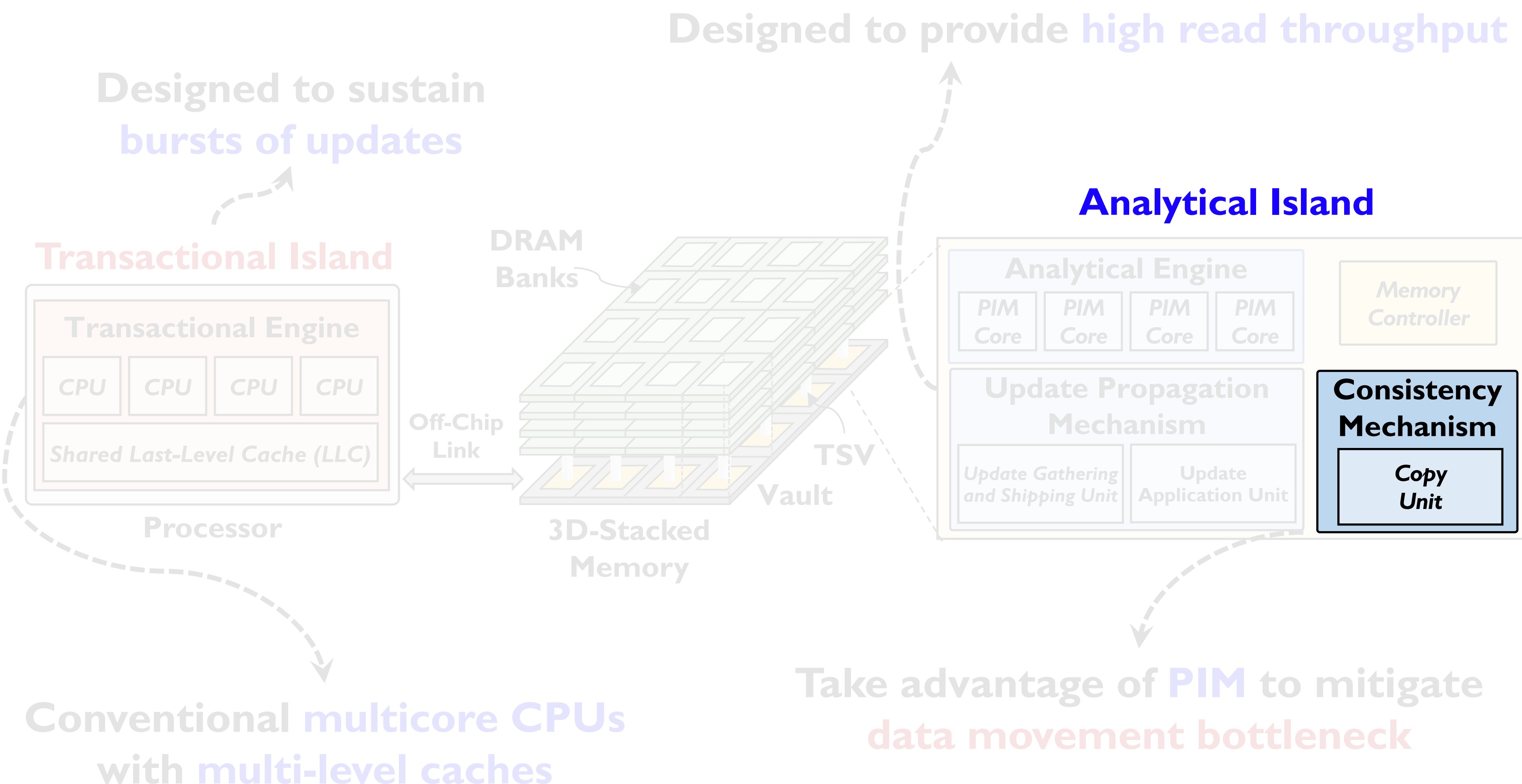
We co-design a new software/hardware accelerator, called update application unit



A 1024-value bitonic sorter, whose basic building block is a network of comparators

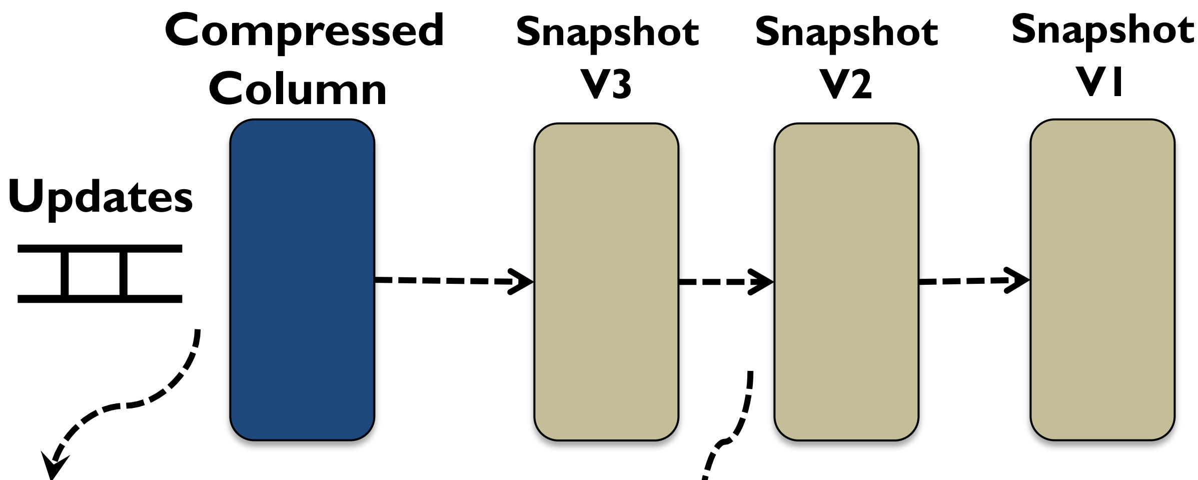
Polynesia: High-Level Overview

Each island includes (1) a replica of data, (2) an optimized execution engine, and (3) a set of hardware resources



Consistency Mechanism

For each column, there is a chain of snapshots where each chain entry corresponds to a version of the column



Polynesia **does not** create a snapshot every time a column is updated. Instead, Polynesia marks the column as **dirty**

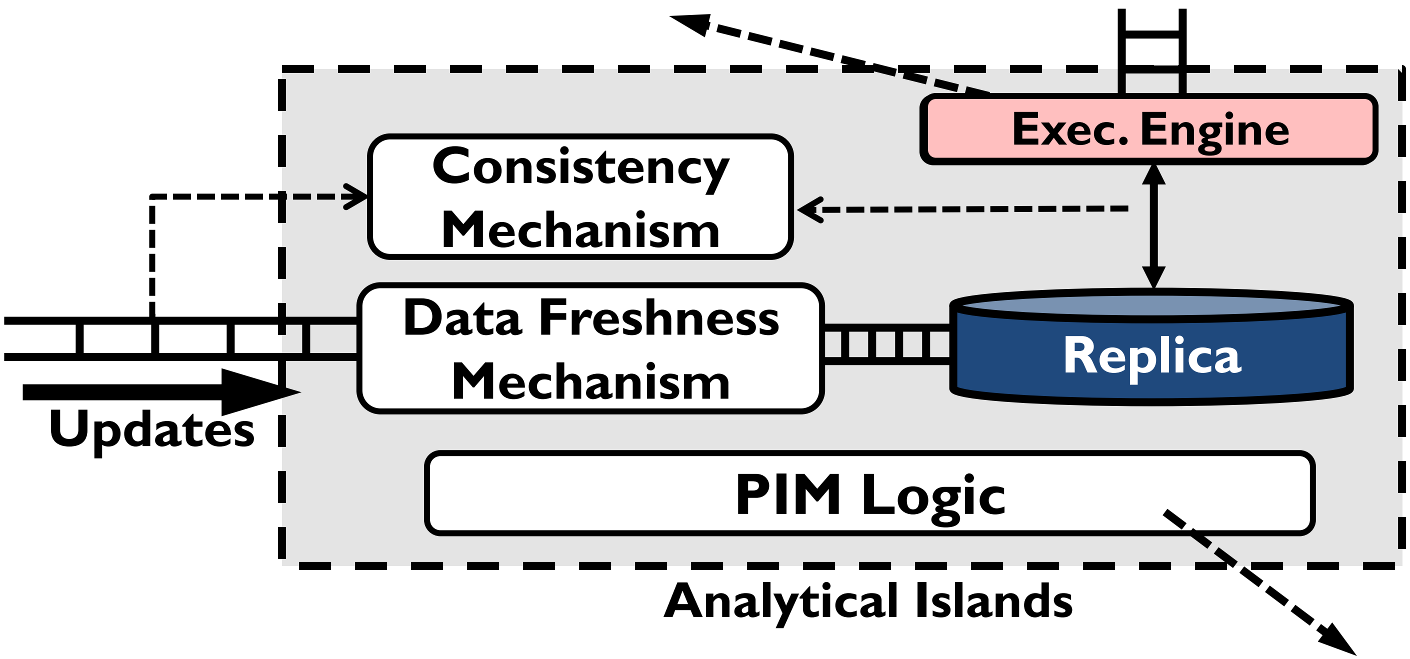
Unlike chains in MVCC, each version is associated **with a column, not a row**

Polynesia creates a new snapshot only if (1) any of the columns are dirty, and (2) no current snapshot exists for the same column

Analytical Islands Key Components

We co-design new algorithms and efficient hardware support for the three key components of an analytical island

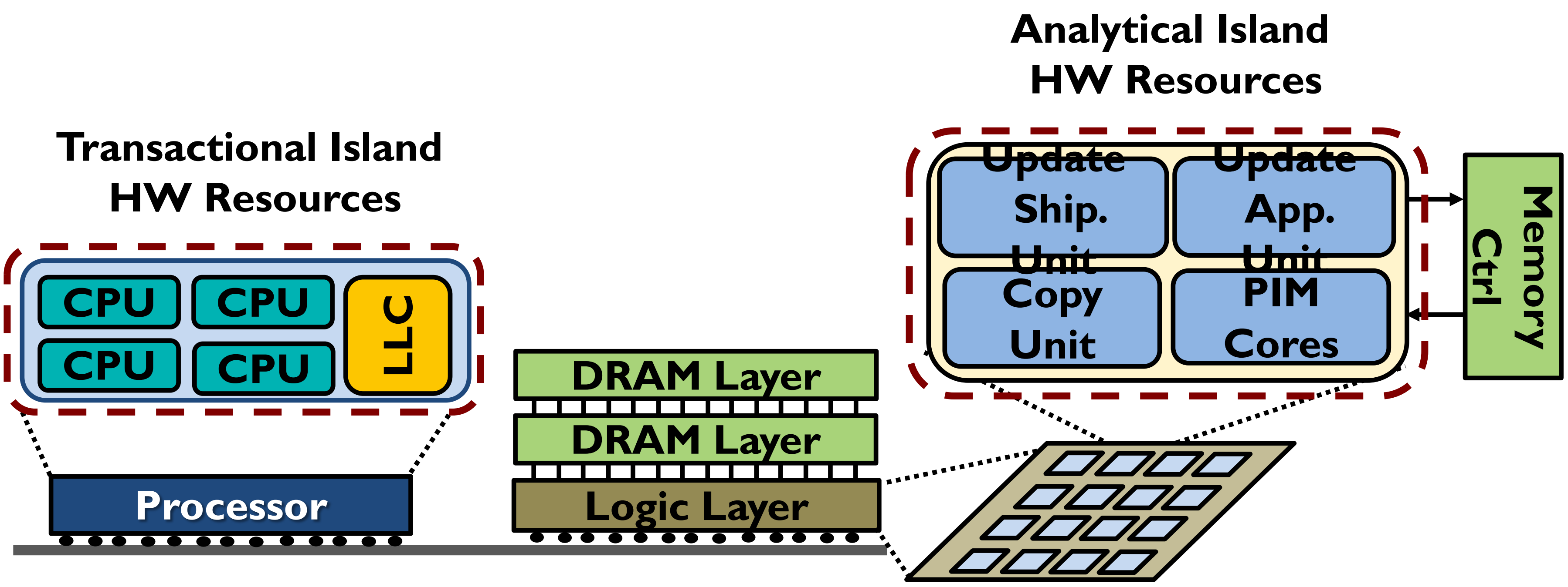
A custom data placement and task scheduler aware of 3D-stacked memory



Simple PIM cores to execute execution engine

A Polynesia HW Implementation

We implement an instance of Polynesia that supports relational transactional and analytical workloads



Consistency Mechanism: Requirements

Consistency mechanism must not compromise either the throughput of analytical queries or the update propagation rate

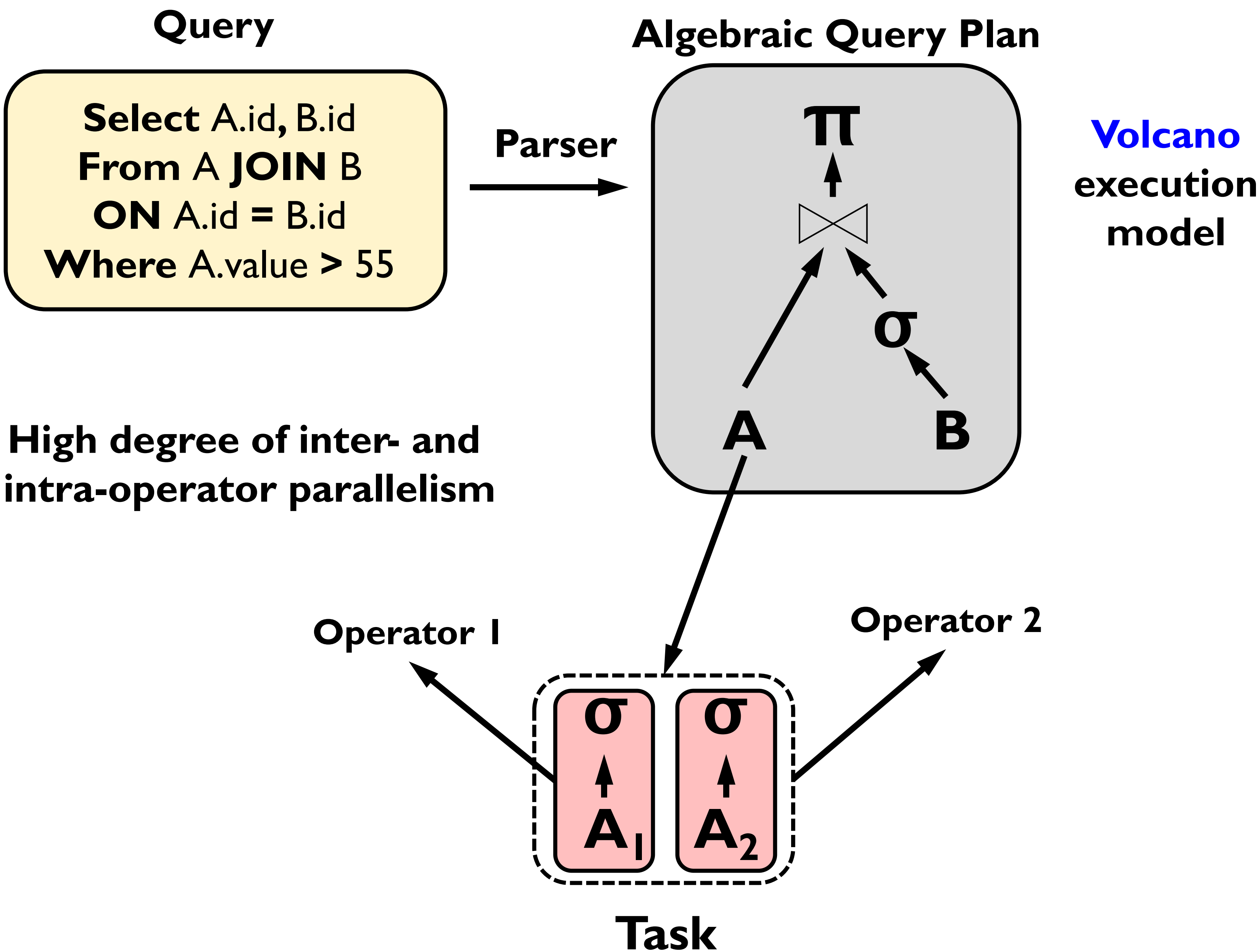
⬇

Consistency mechanism has to satisfy two requirements:

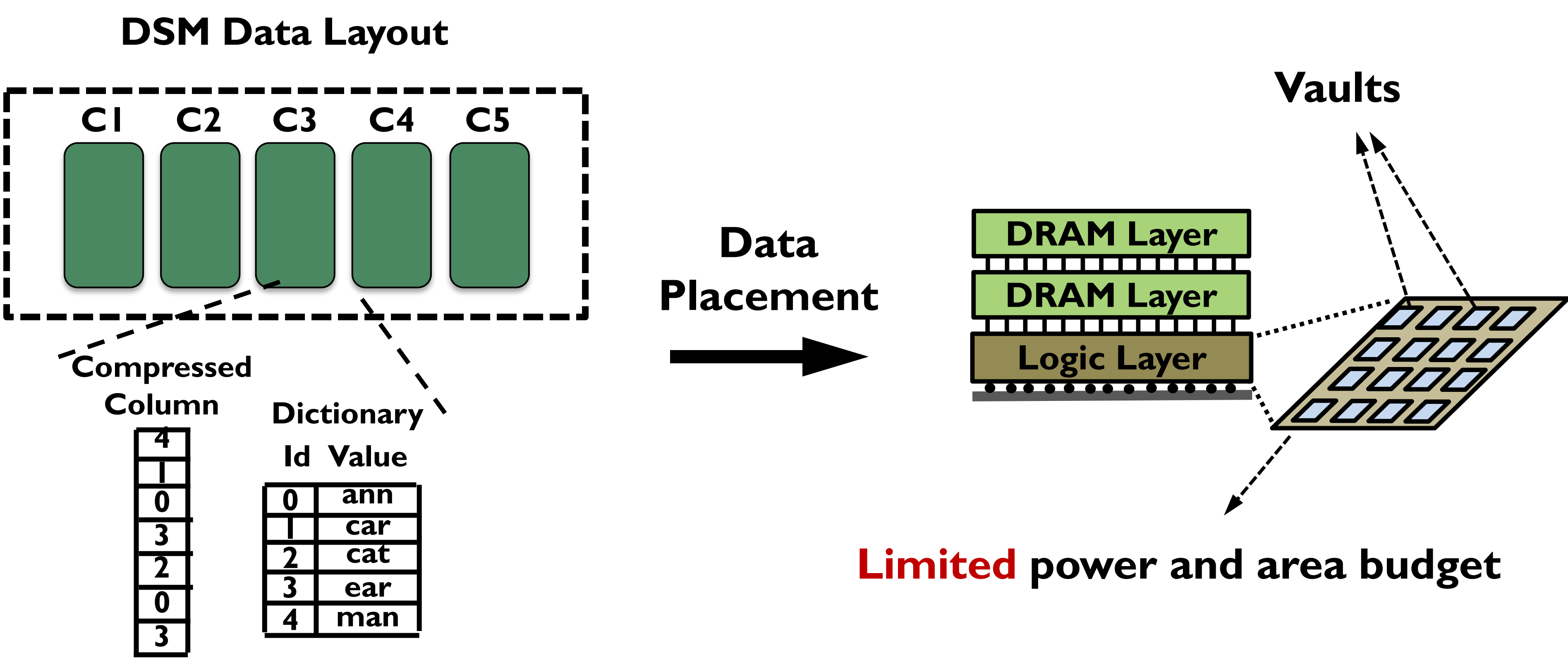
1. **Updates must be applied all the time and should not be blocked by analytical queries → Data freshness property**

2. **Analytics must be able to run all the time and should not be blocked by update propagation process → Performance isolation property**

Analytical Engine: Query Execution



Analytical Engine: Data Placement



Analytical Engine: Task Scheduler

For each query, the scheduler makes three key decisions:

1 **Decides how many tasks to create**

2 **Finds how to map these tasks to the available resources (PIM threads)**

3 **Guarantees that dependent tasks are executed in order**

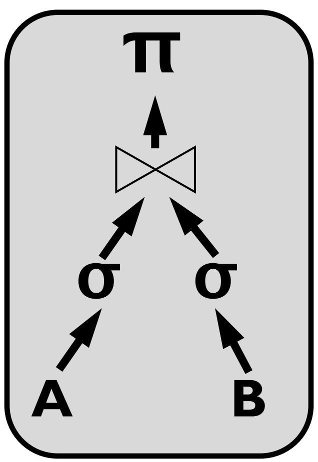
Task Scheduler: Initial Hueristic

Our scheduler heuristic that generates tasks by disassembling the operators of the query plan into operator instances

Query

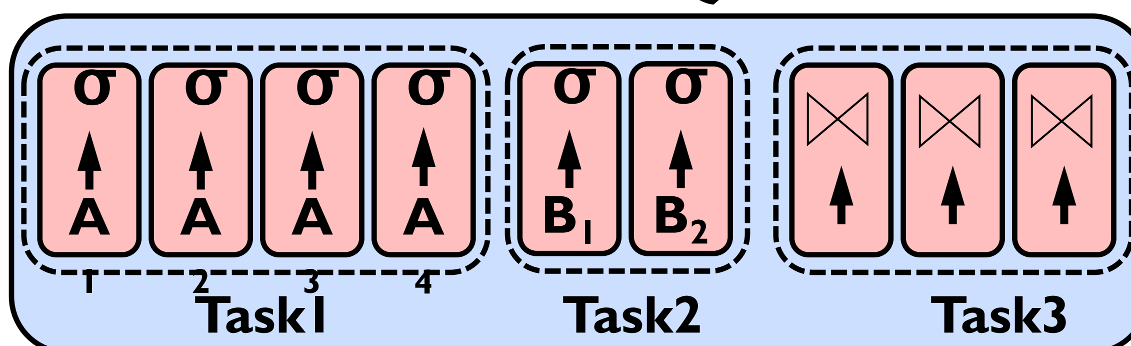
Select A.id**,** B.id
From A **JOIN** B
ON A.id **=** B.id
Where A.value **>** 55
Where B.value **<** 70

Query Plan



Scheduler

Global Work Queue



(1) which vault groups the input tuples reside in, (2) the number of available PIM threads in each vault group

Task Scheduler: Initial Heuristic

We find that this heuristic is not optimized for PIM and leads to sub-optimal performance due to three reasons:

1 **The heuristic requires a dedicated runtime component to monitor and assign tasks**

- **The runtime component must be executed on a general-purpose PIM core**

2 **The heuristic's static mapping is limited to using only the resources available within a single vault group**

- **Can lead to performance issues for queries that operate on very large columns**

3 **This heuristic is vulnerable to load imbalance**

- **Some PIM threads might finish their tasks sooner and wait idly for straggling threads**

Task Scheduler: Optimized Hueristic

We optimize our heuristic to address these challenges:

1 **We design a pull-based task assignment strategy, where PIM threads cooperatively pull tasks from the task queue at runtime**

- **We introduce a local task queue for each vault group**
- **This eliminates the need for a runtime component (first challenge) and allows PIM thread to dynamically load balance (third challenge)**

2 **We optimize the heuristic to allow for finer-grained tasks**

- **Partition input tuples into fixed-size segments (i.e., 1000 tuples) and create an operator instance for each partition**

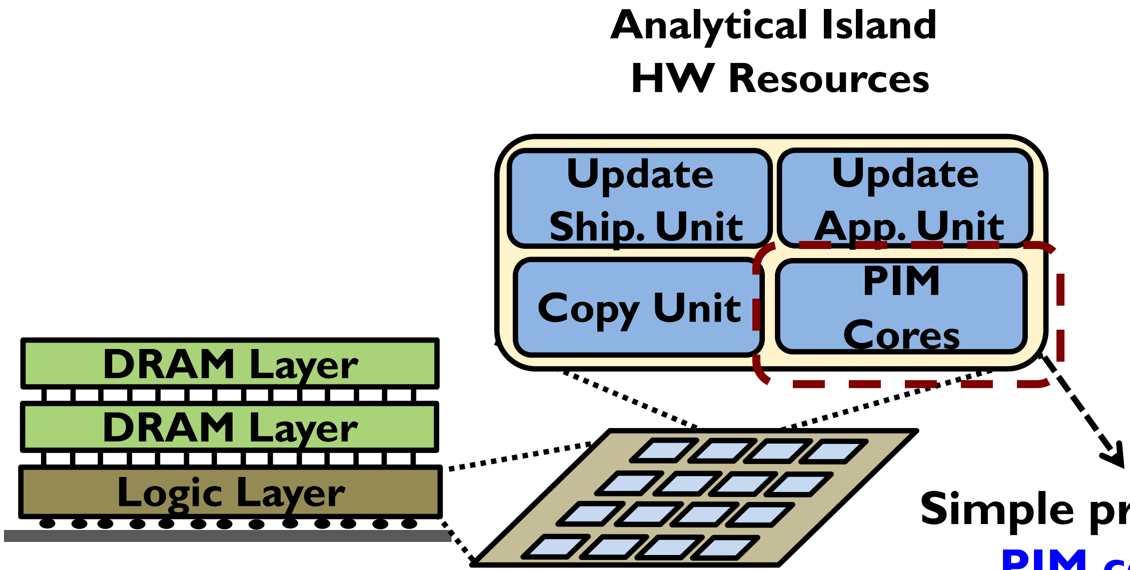
3 **We optimize the heuristic to allow a PIM thread to steal tasks from a remote vault if its local queue is empty**

- **This enables us to potentially use all available PIM threads to execute tasks**

Analytical Engine: Hardware Design

Given area and power constraints, it can be difficult to add enough PIM logic to each vault to saturate the available vault bandwidth

Our new data placement strategy and scheduler enables us to expose greater intra-query parallelism



Simple programmable in-order PIM cores to exploit the available vault bandwidth

Wrap up: Single-Instance Systems

While single-instance design enables high data freshness, we find that it suffers from two major challenges:

1 **High Cost of Consistency and Synchronization**

2 **Limited Performance Isolation**

Consistency Mechanism: Algorithm

Our mechanism relies on a combination of snapshotting and versioning to provide snapshot isolation for analytics

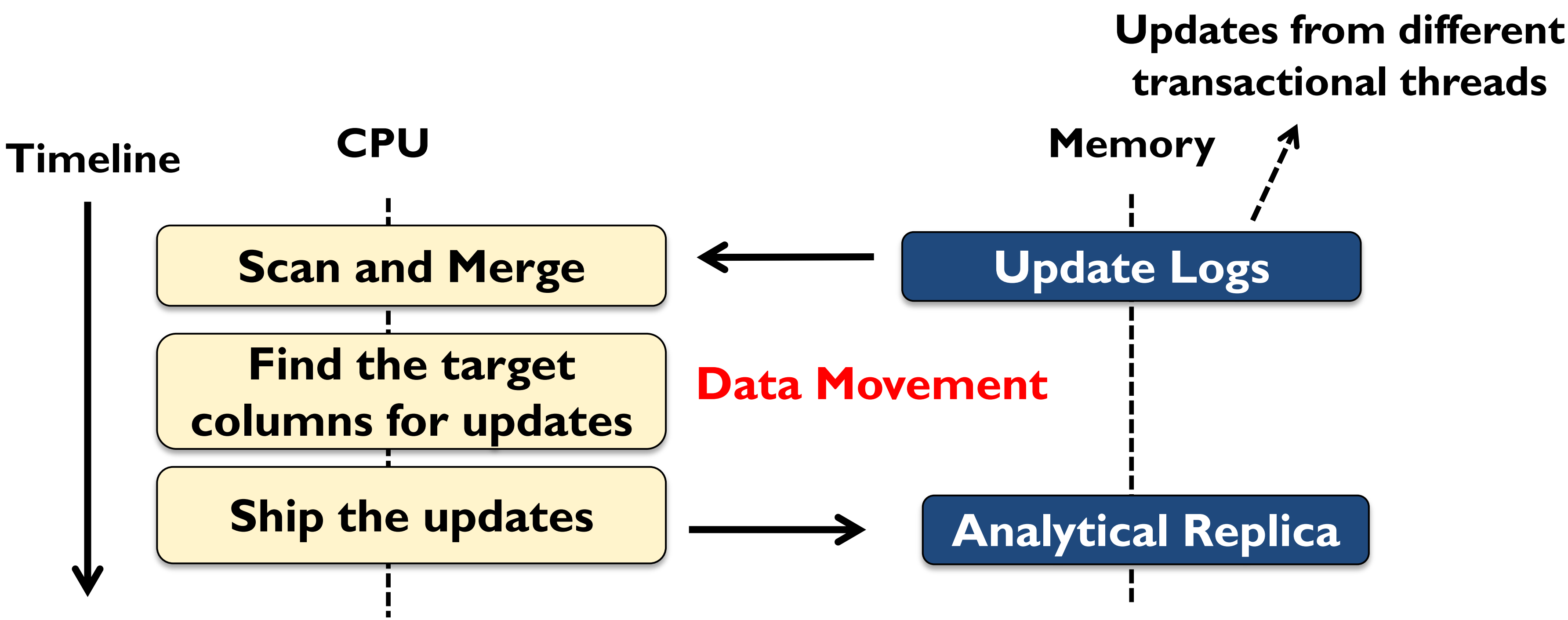
Our consistency mechanism is based on two key observations:

1 Updates are applied at a column granularity

2 Snapshotting a column is cost effective using PIM

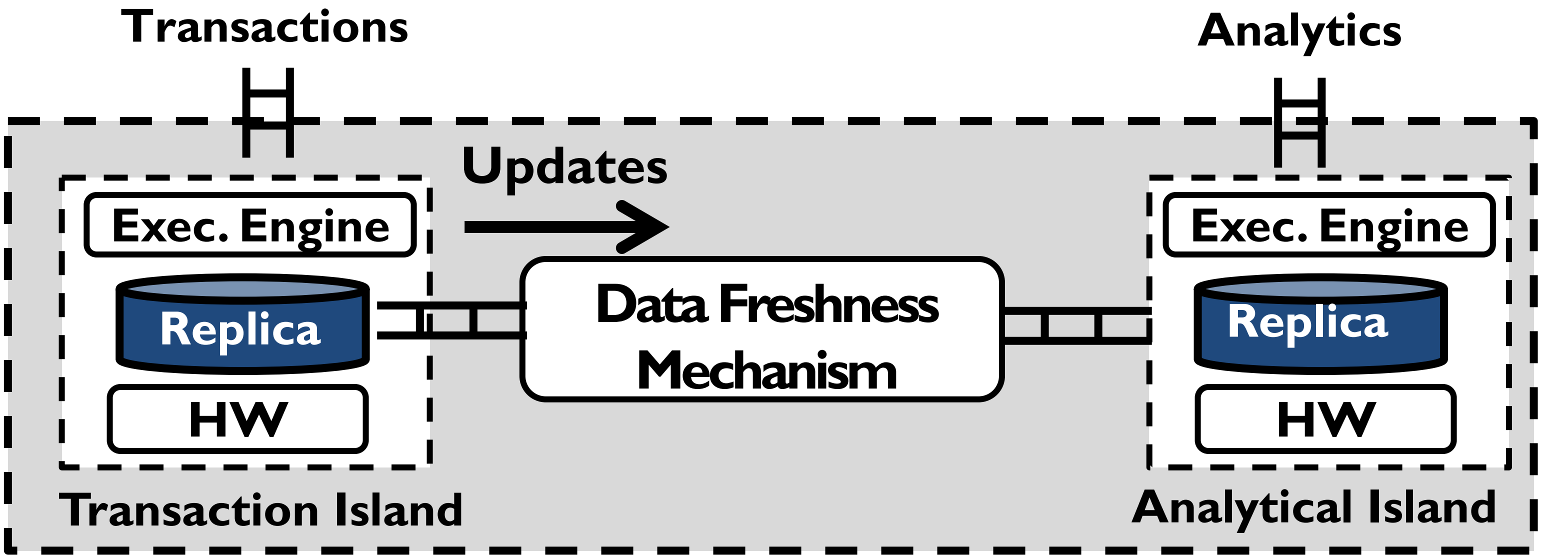
Update Propagation: Update Gathering & Shipping

Goal: **gather updates from transactional threads and ship them to analytical the replica**



High update rate → Frequent update gathering & shipping → Higher data movement overhead

Data Freshness Mechanism



Data Freshness Mechanism:

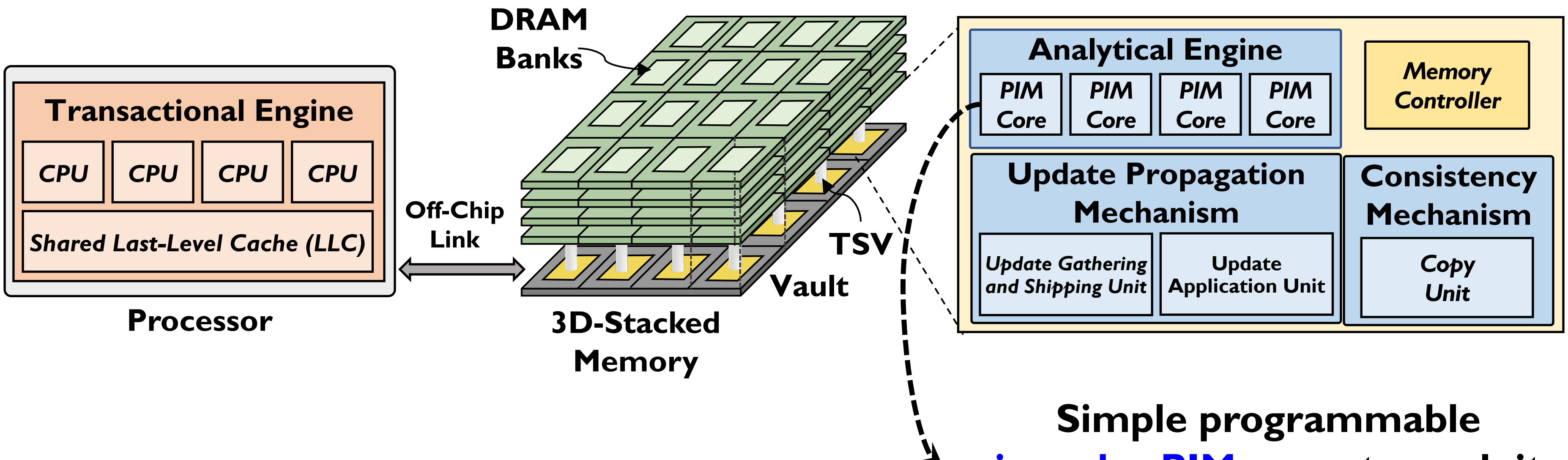
1 **Update Shipping:** gather updates from transactional islands, find the target location in analytical island, and ship them

2 **Update Application:** performs format conversion and applies the update to the analytical replica

Analytical Engine: Hardware

Given area and power constraints, it can be difficult to add enough PIM logic to each vault to saturate the available vault bandwidth

Our new data placement strategy and scheduler enables us to expose greater intra-query parallelism



Simple programmable in-order PIM cores to exploit the available vault bandwidth

Analytical Engine: Query Execution

Efficient analytical query execution strongly depends on:

1 Data layout and data placement

2 Task scheduling policy

We design **a pull-based** task assignment strategy, where **PIM** threads **cooperatively** pull tasks from the task queue **at runtime**

3 How each physical operator is executed

We employ the **top-down Volcano (Iterator)** execution model to execute physical operations (e.g., scan, filter, join) while respecting operator's dependencies