PA152: Efficient Use of DB

12. New SQL

Vlastislav Dohnal

Credits

- This presentation is based on:
  - S. Harizopoulos et al: *OLTP Through the Looking Glass, and What We Found There.* SIGMOD, 2008
  - M. Stonebraker et al: *H-Store: The End of an Architectural Era.* VLDB, 2007
  - J. DeBradant et al: *Anti-Caching: A New Approach to Database Management System Architecture*. VLDB, 2013.

Contents

- Features of OLTP
- Trends in OLTP
- Performance study of individual bottlenecks
- H-store

Main Features of OLTP

- **Buffer Management**
  - to facilitate data transfer between memory and disk
- **B-Tree** for on-disk data storage
- **Logging** for recovery
- **Locking** to support concurrency
- **Latching** for accessing shared data structure

Motivation

- Is the OLTP database optimized nowadays, given the hardware advancement?
- Request from outside the DB community for alternative DB architecture

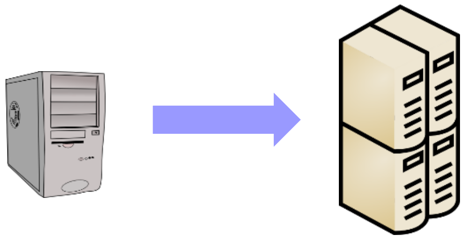
Motivation: Hardware advancement

	In 1980s	Nowadays
HW cost	In millions	Few thousands
Storage size	DB size >> Memory	Memory > DB size
Processing time for most of the transactions	\	In microseconds

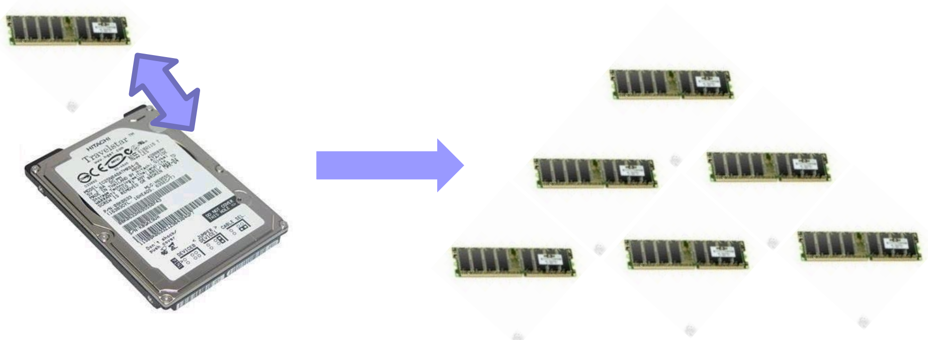
Motivation: Request from outside

- "Database-like" storage system proposal from Operating System and networking conference
  - varying forms of
    - concurrency,
    - consistency,
    - reliability,
    - replication,
    - queryability.

Trends in OLTP



1. Cluster computing
2. Memory resident databases
   - Data in OLTP doesn’t grow as fast as memory size.



3. Single threading
4. High availability vs. Logging
5. Transaction Variants

Trend 3: Single Threading

- A step backward from multithread to single thread?
- Why multithreading?
  - Prevent idle of CPU while waiting data from disk
  - Prevent long-running transactions from blocking short transaction
- Not valid for memory resident DB
  - No disk wait
  - Long-running transactions run in warehouse

Trend 3: Single Threading

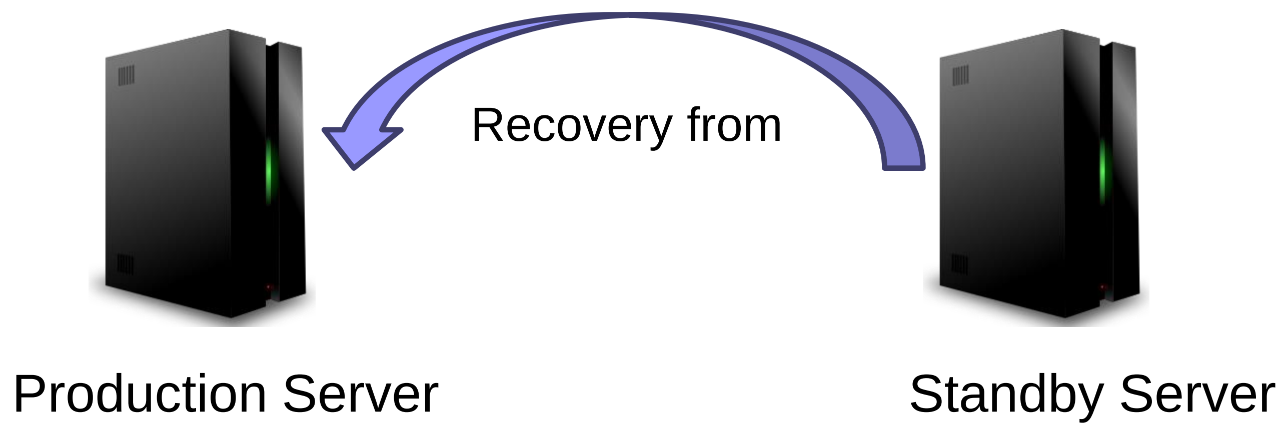
- What about multi processors?
  - Dynamic locking was experimentally the best concurrency control with disk.
  - What concurrency control protocol is best?
- Goal: Achieve shared-nothing processor by virtual machine
  - So, concurrency control code gets removed.

Trend 3: Single Threading

- What about network disk?
  - Feasible to partition transaction to run in “single-site”.
  - Intra-query parallelism: each processor running on a part of a single query.

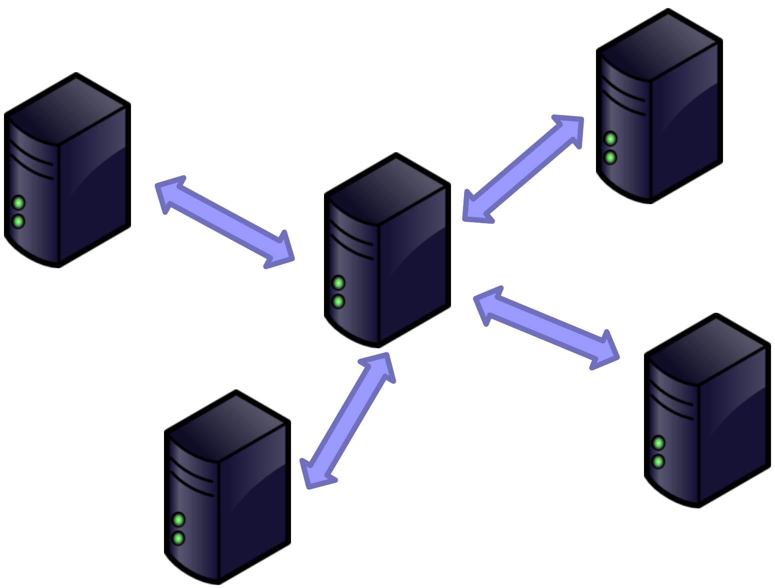
Trend 4: HA vs. Logging

- 24x7 service achieved by using multiple sets of hardware.
- Perform recovery by copying missing states from other database replicas.
  - Log for recovery can be avoided.



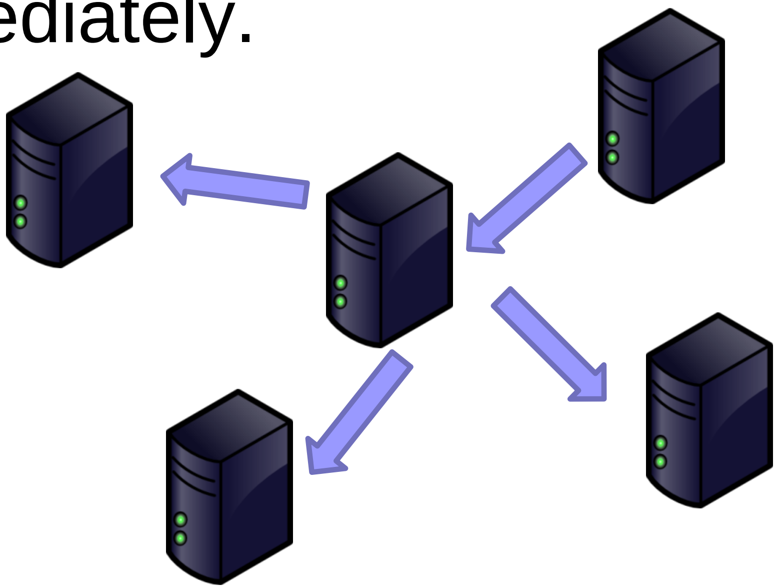
Trend 5: Transaction Variants

- Why transaction variants?
  - 2-phase commit protocol harm performance of large-scale distributed DB system
  - 2-phase commit involves *commit-request and commit phase* which involves all server to participate.

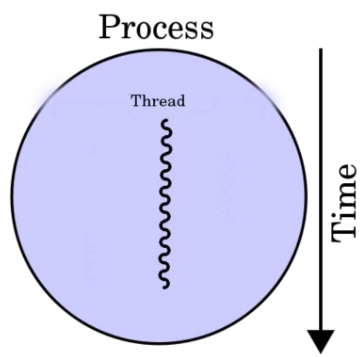


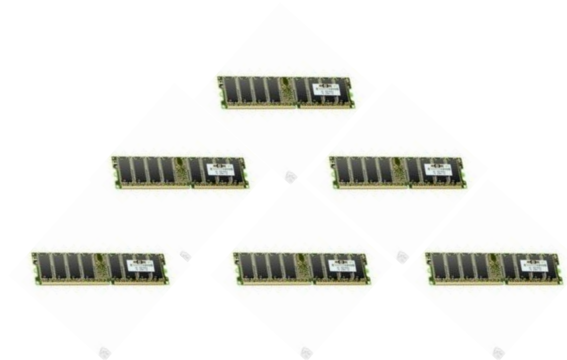
Trend 5: Transaction Variants

- Trade consistency for performance
- Eventual consistency, all writes propagate among the database servers.
  - But not immediately.



Trend in OLTP - Summary

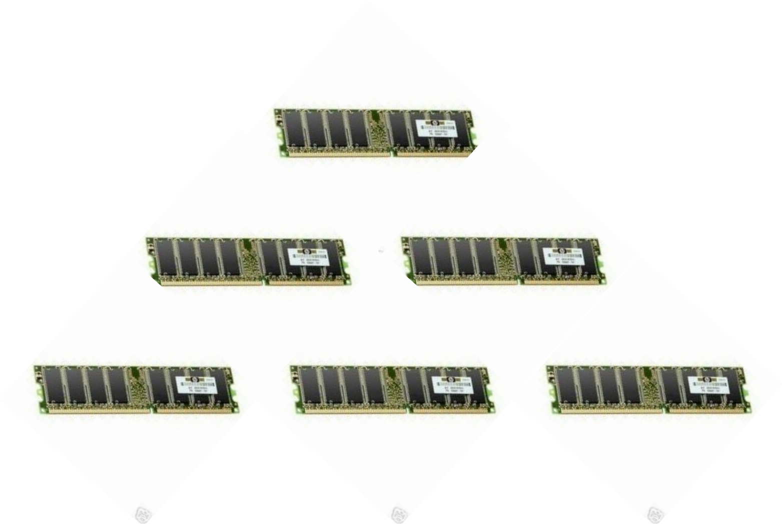




Eventual consistency

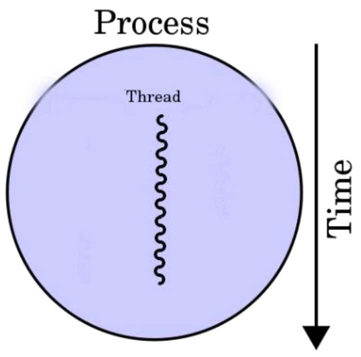
Impact on DBMS

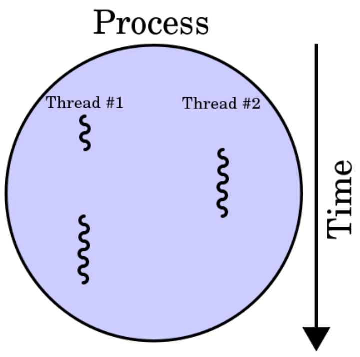
- (1) memory resident DB can get rid of **buffer management**



Impact on DBMS

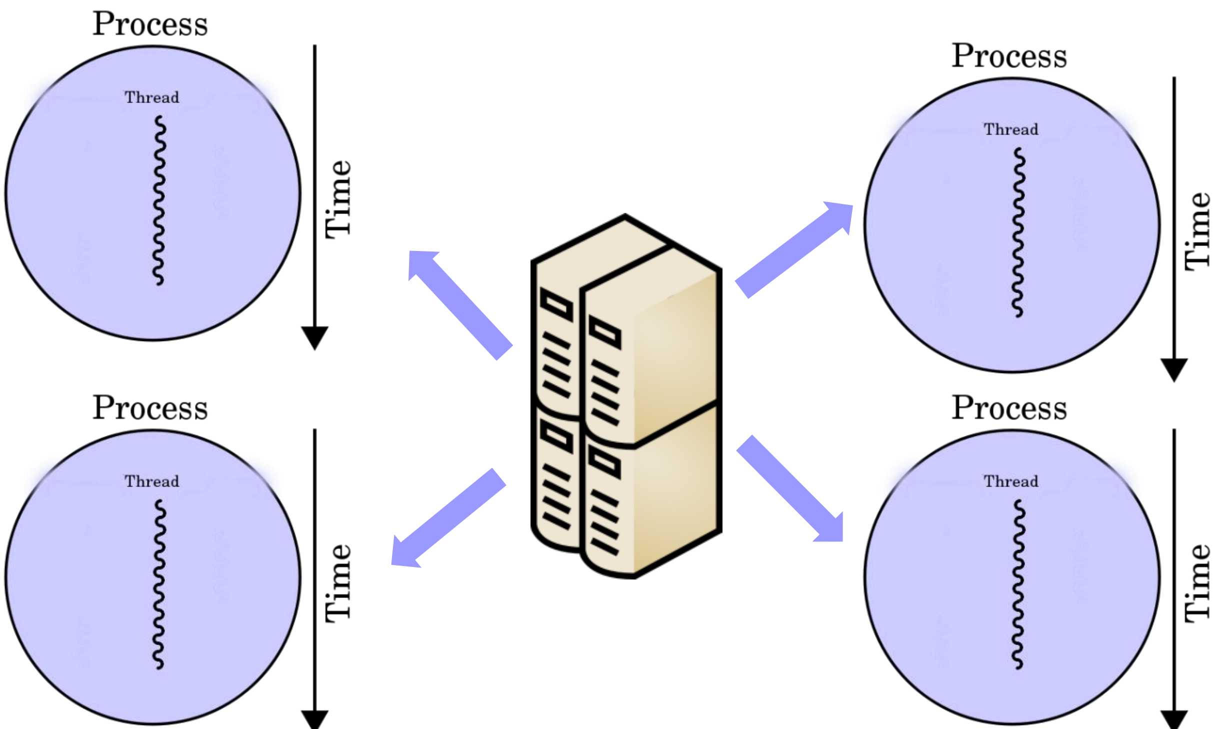
- (2) single thread can avoid **locking** and **latching**





Impact on DBMS

- (3) cluster computing helps avoid **locking.**
  - Instead of single processor and multithreading, each processor is responsible for each own thread.



Impact on DBMS

- (4) high availability without replication mgr.
  - Active-passive replication scheme (log shipping)
    1. Replica may not be consistent with the primary
       - unless on two-phase commit protocol
    2. Failover in not instantaneous
    3. Log is required
       - It takes about 20% of CPU cycles.
  - Active-active replication scheme with transactions
    - Two-phase commit introduces large latency for distributed replication yet.

Impact on DBMS

- (5) being “transaction less” avoids book keeping, i.e., **logging.**

- (5+) Cache-conscious B-Trees
  - Cache misses in the B-tree code may well be the new bottleneck for the stripped-down system.
    - Related to utilization of the first-level data cache of the CPU.

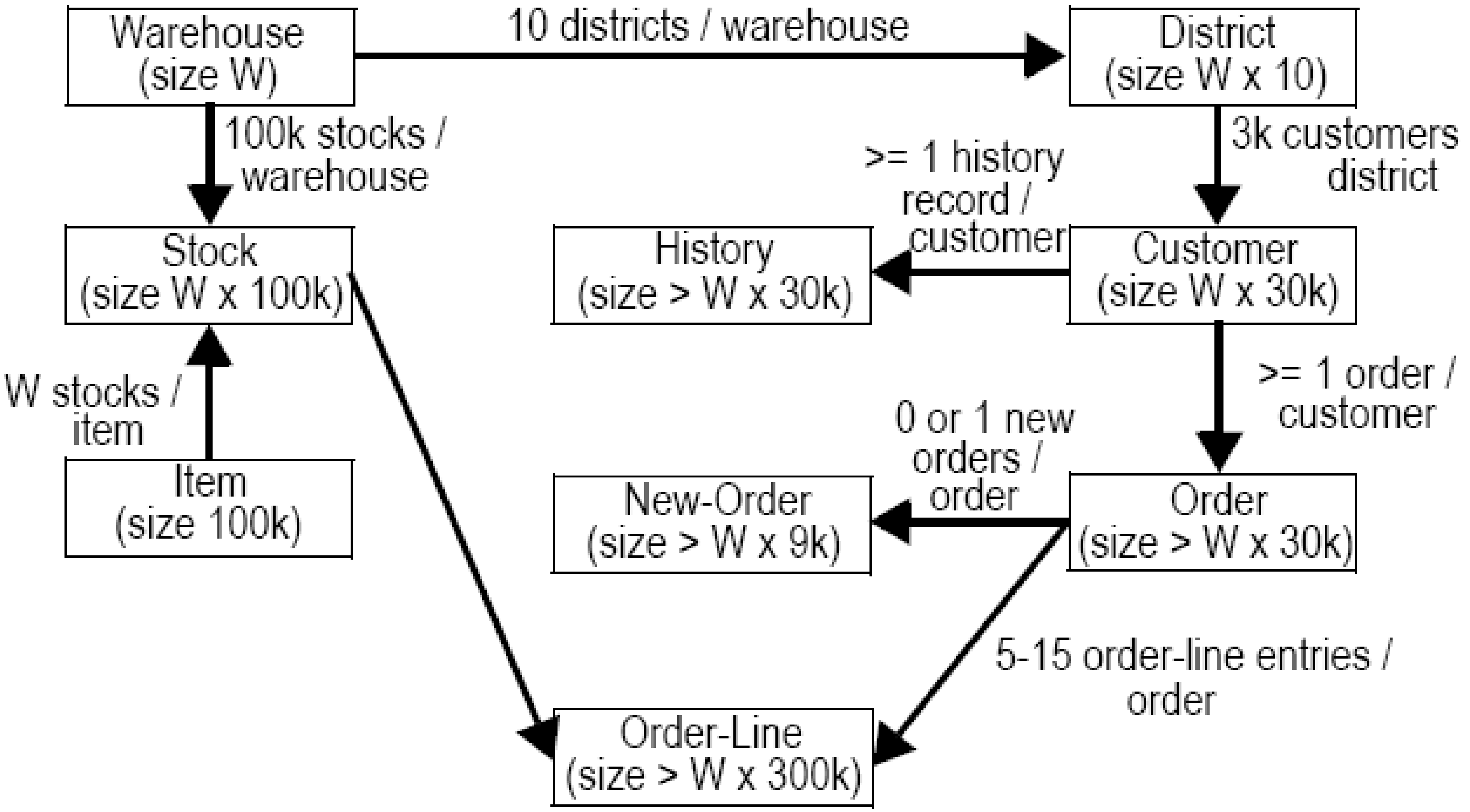
TPC-C Benchmark

http://www.tpc.org/tpcc/default.asp

- TPC-C is industry standard used to measure ecommerce performance
- TPC-C is designed to represent any industry that must manage, sell, or distribute a product or service
- Vendors includes Microsoft, Oracle, IBM, Sybase, Sun, HP, DELL etc.

TPC-C Benchmark

- 1 warehouse(~100M) serves 10 districts, and each district serve 3000 customers.



TPC-C Benchmark

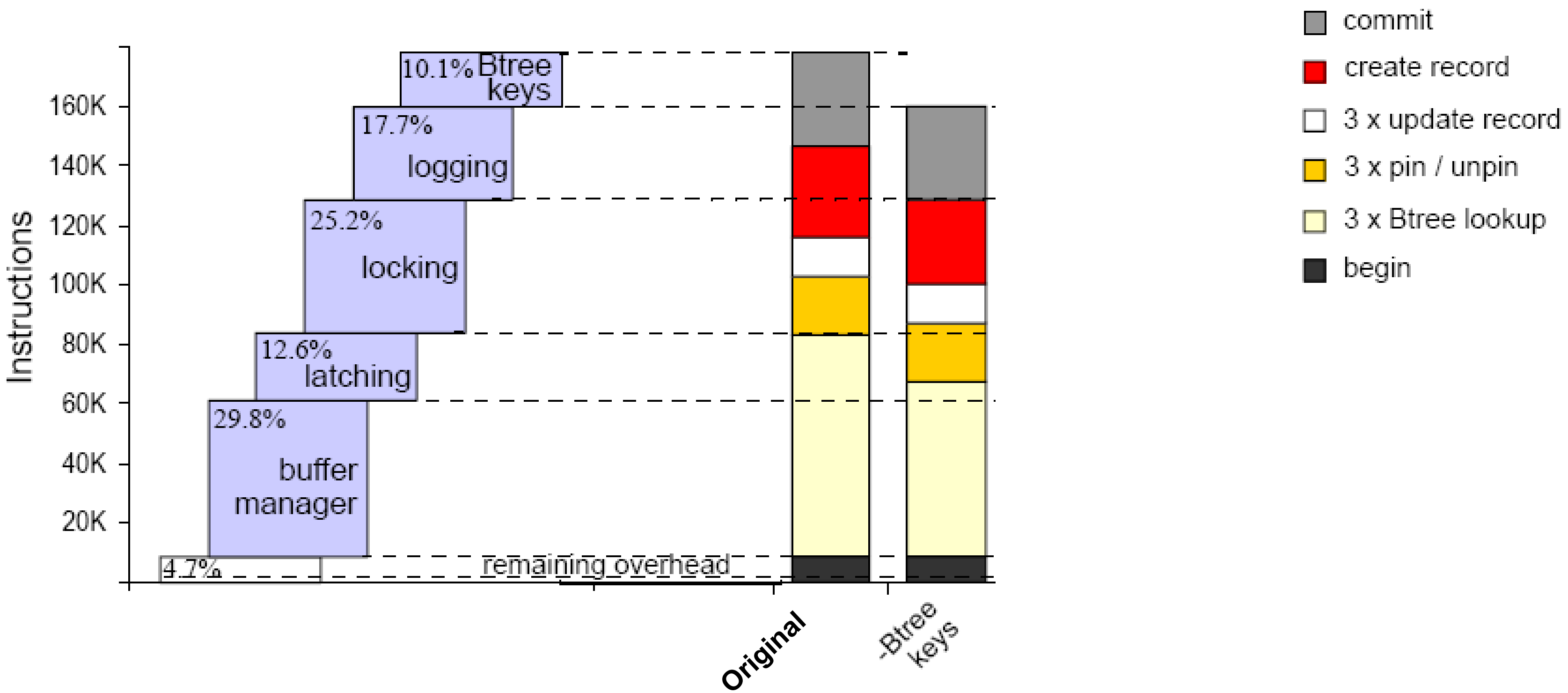
- 5 concurrent business transactions
  - New Order Transaction
  - Payment
  - Deliver Order
  - Check status of Order
  - Monitor Stock Level of warehouse

Experiment setup

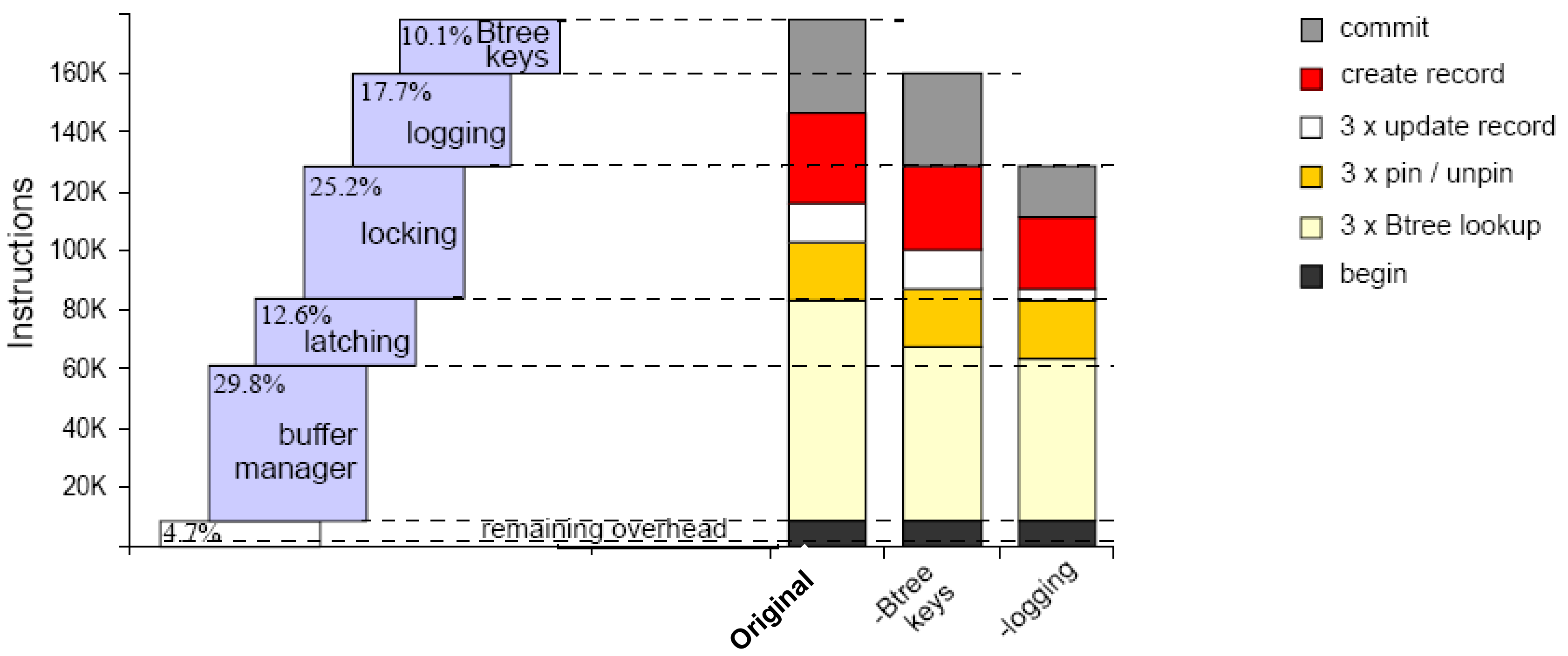
- 40,000 transactions run for types
  - *New Order Transaction* and *Payment*
- Results measured in
  - Throughput (Time, Transactions completed)
  - Instruction count
- Single-core Pentium 4, 3.2GHz, with 1MB L2 cache, hyper threading disabled, 1GB RAM, running Linux 2.6.

Effect of removing components (1)

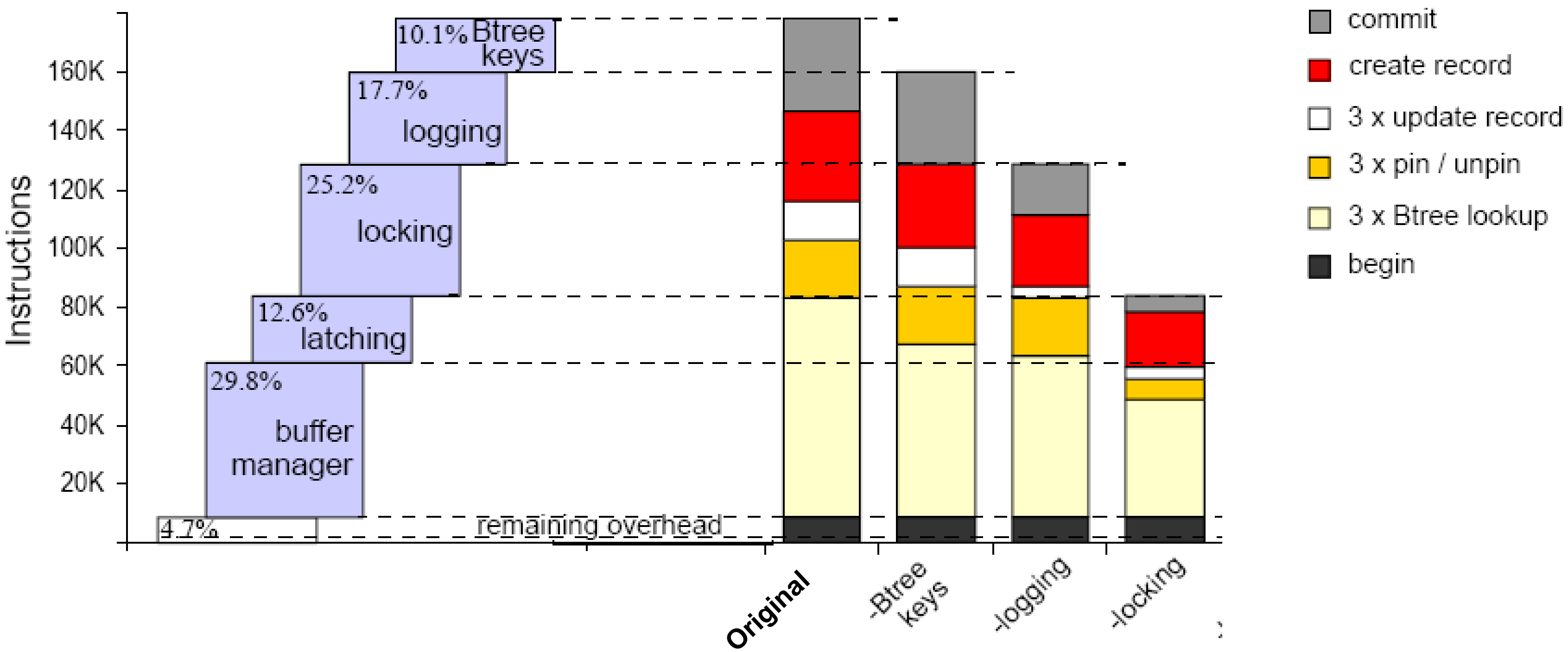
- Payment transaction:



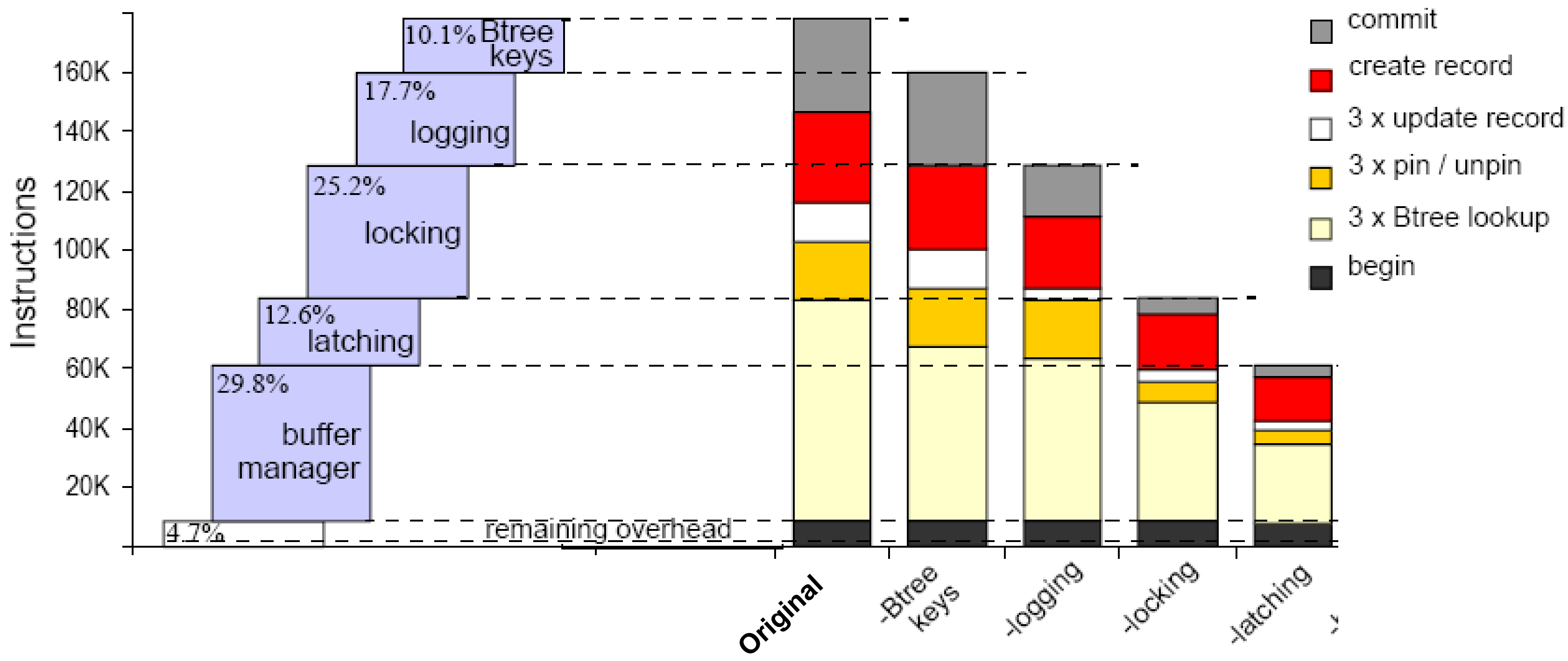
Effect of removing components (2)



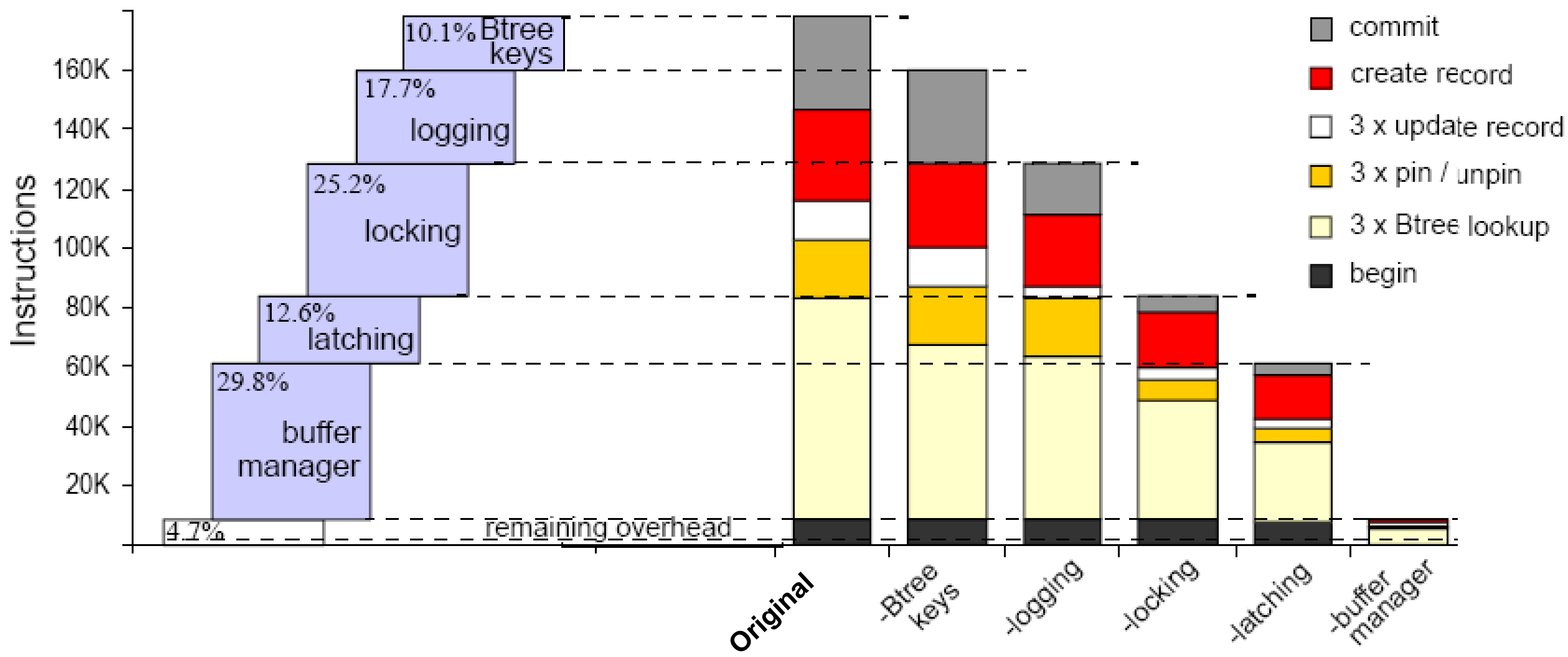
Effect of removing components (3)



Effect of removing components (4)

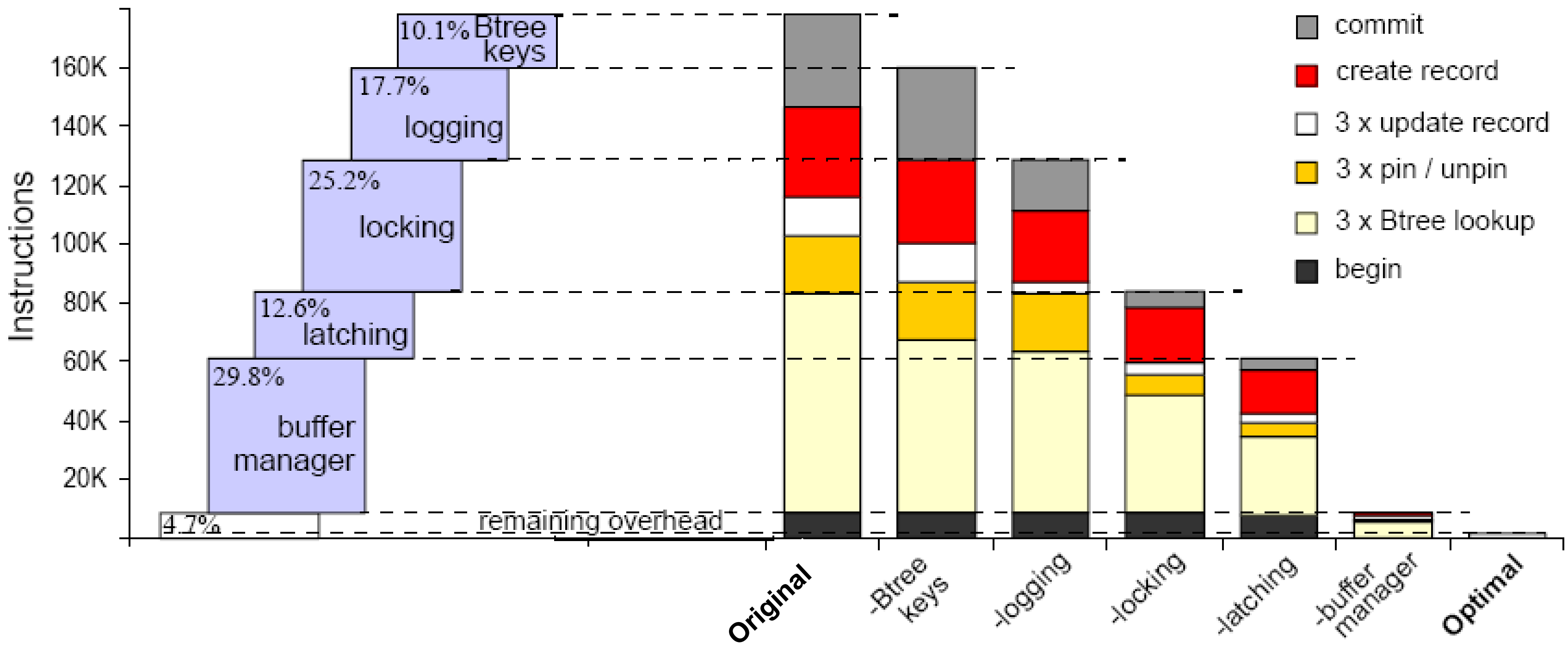


Effect of removing components (5)



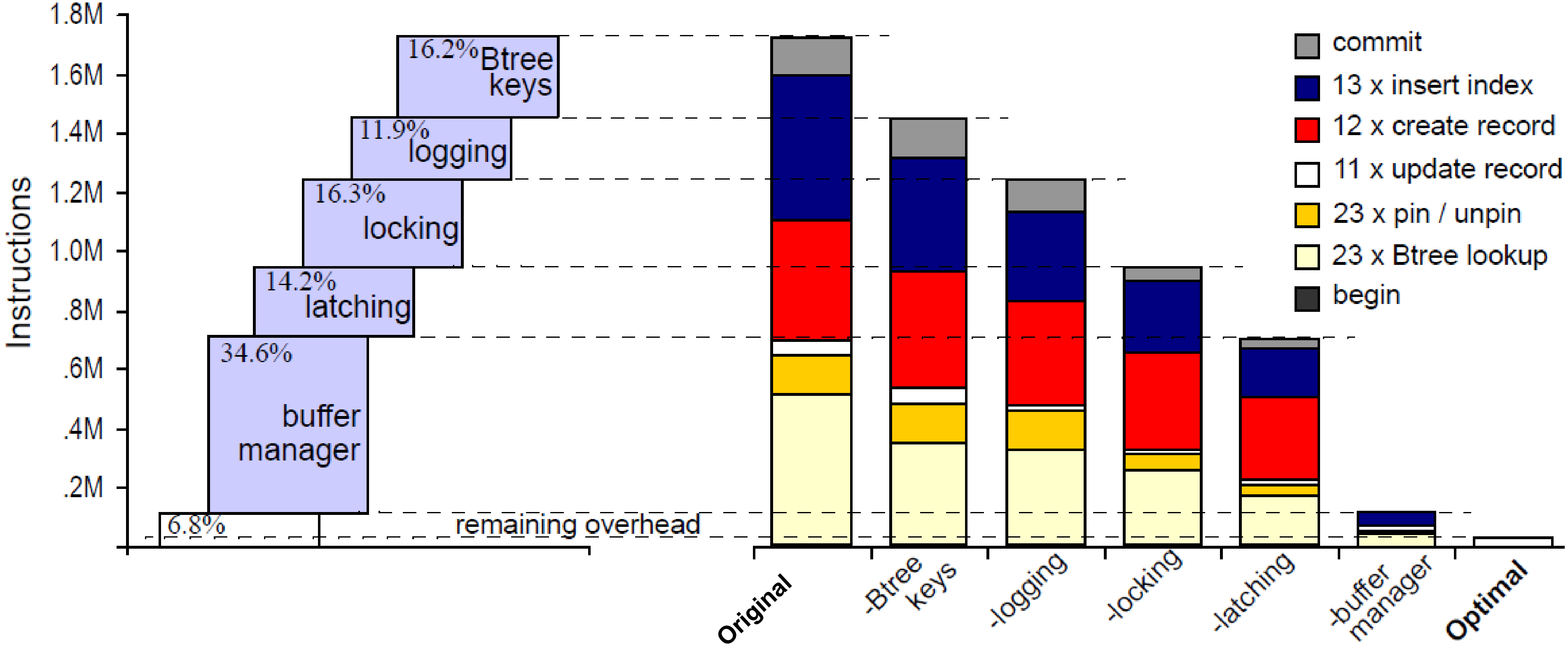
Effect of removing components (6)

- Instructions of useful work is only <2% of a memory resident DB



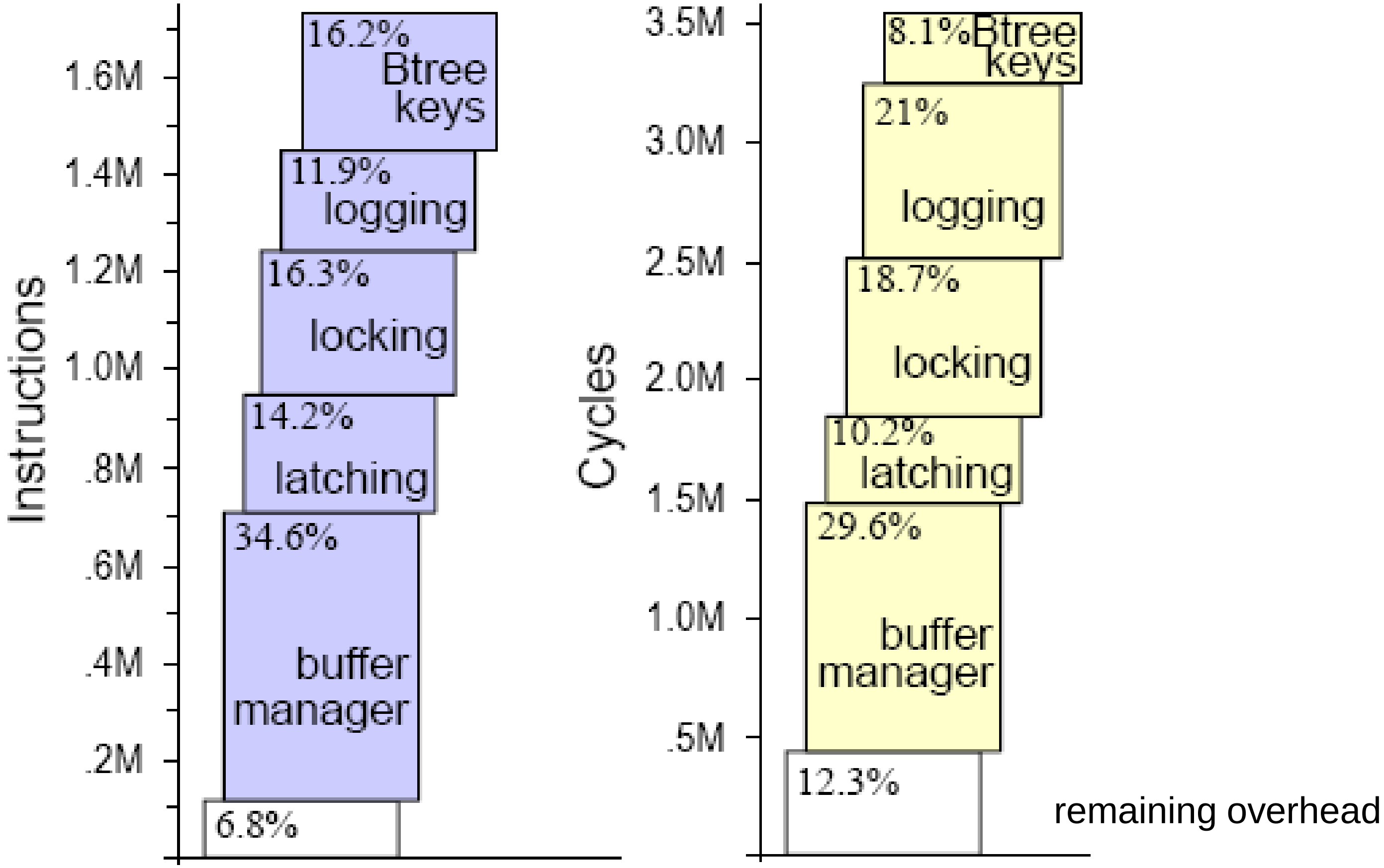
Effect of removing components (7)

- The same for New Order Transaction



Effect of removing components (8)

- Comparison of CPU instructions and cycles
  - New order transaction



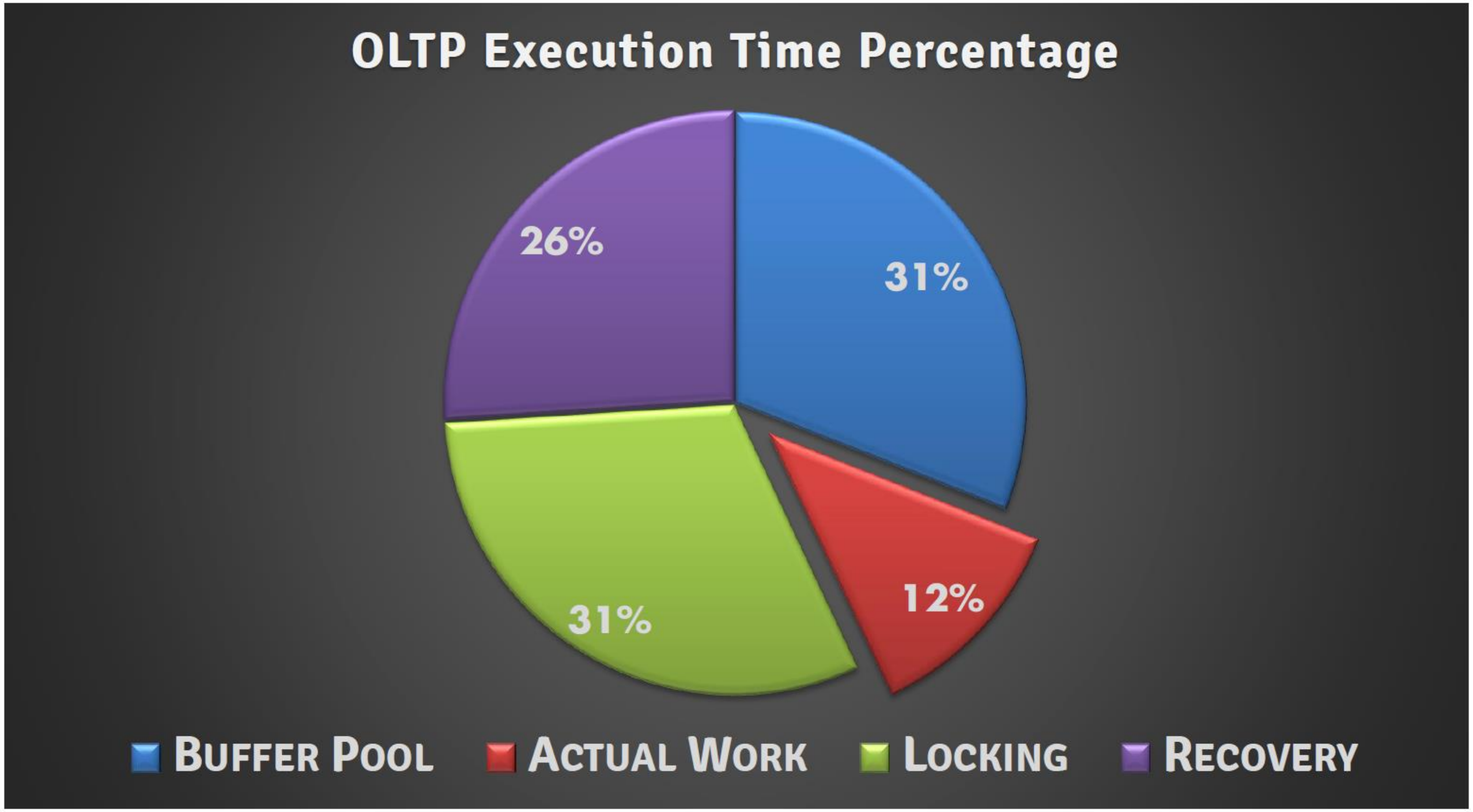
Experiment Results

- Memory resident DBMS
  - 640 transactions per second.
- Stripped-down DBMS
  - 12,700 transactions per second.
- Stripped-down DBMS gave a 20 times improvement in throughput

Conclusion

- Most significant overhead contributors
  - buffer management and
  - locking operation,
  - followed by logging and latching.
- A fully stripped-down system's performance is orders of magnitude better than an unmodified system.
  - "One size fits all" DBMSes excel at nothing
    - Need for specialized databases and languages

Conclusion



© OLTP Through the Looking Glass, and What We Found There, SIGMOD '08

- Welcome to NewSQL

NewSQL DBMS

- Highly concurrent, latch-free data structures
- Partitioning into single-threaded executors

- H-store
  - Distributed, shared-nothing, main mem DBMS
  - Row-store based relational DBMS

ZDNet interview, Feb. 2008

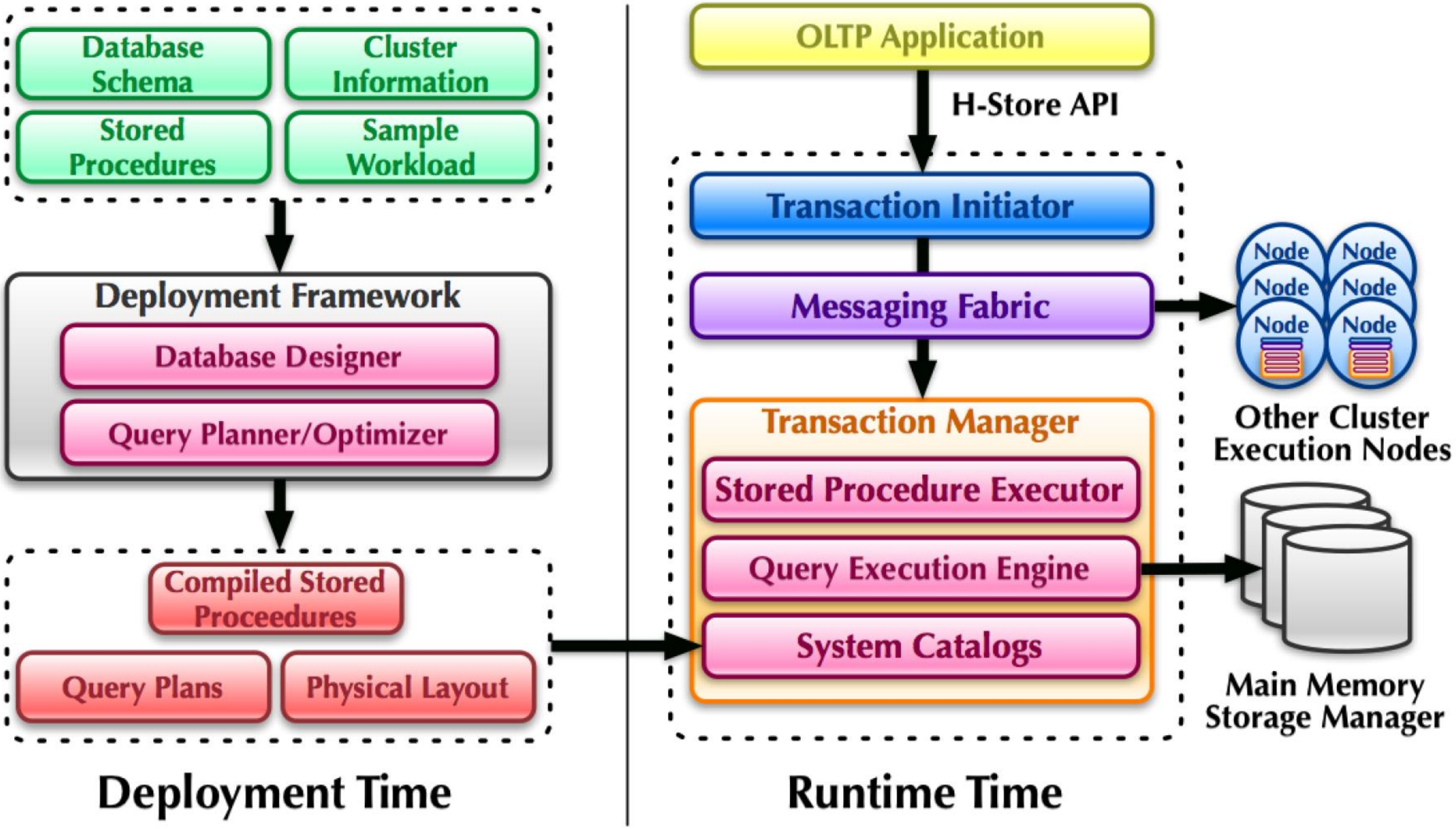
Is H-Store going to be a complete replacement for Oracle? No. Oracle does lots of things, and replacing it requires a variety of more specialized technologies. Stonebraker and I have been going back and forth about the exact list, but it's something like:

- High-end OLTP (Oracle, SQL Server, DB2 today – eventually H-Store
- Mid-range OLTP (MySQL, PostgreSQL, EnterpriseDB, Progress)
- Row-based analytic (Teradata, Netezza, DATAllegro)
- Column-based analytic (Vertica, ParAccel, Infobright) – these also win for RDF
- Scientific
- Text and XML (Microsoft/FAST, Autonomy, Google, Coveo, Marklogic, Attivio)
- Embedded (SQL Anywhere, solidDB)
- Stream non-DBMS (Coral8, StreamBase, Apama)
- Big cloud sub-DBMS (MapReduce, Hadoop, SimpleDB)

H-Store

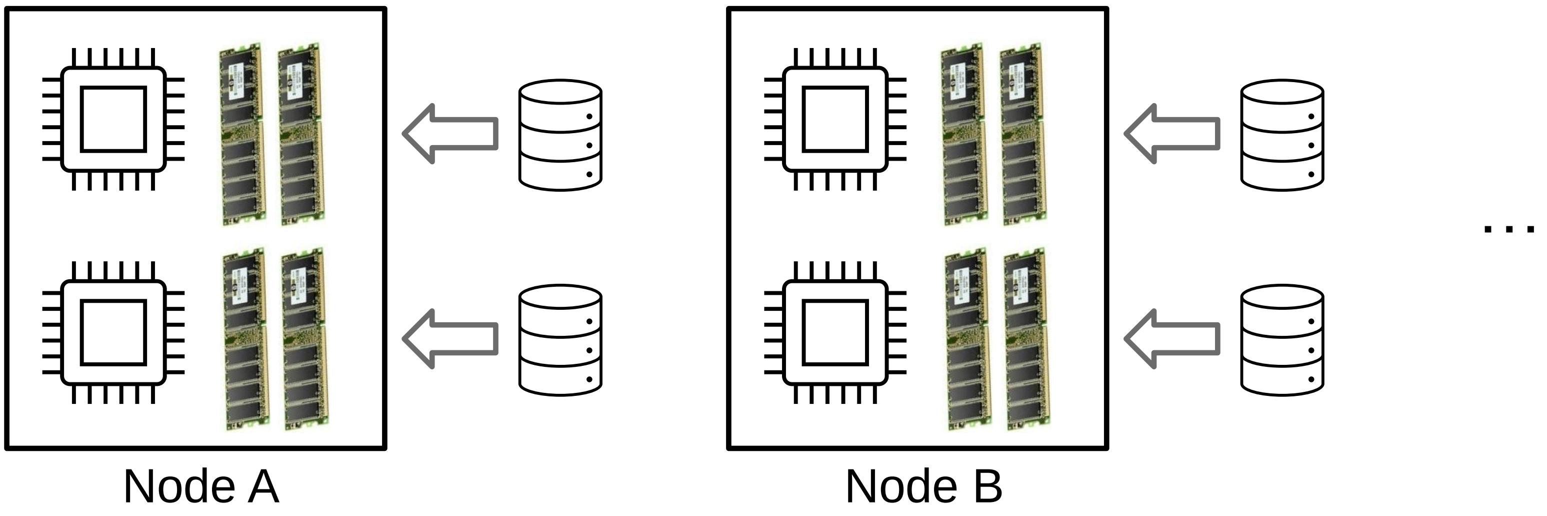
- Logging overhead
  - Replication for recovery → no redo log
  - Transient undo log sufficient for tx rollback
- Transaction classes
  - Optimize concurrency control protocols
- Incremental scalability
  - Shared nothing architecture
- Remove knobs/tuning parameters
  - Personnel costs higher than machine costs
  - Automatic physical database design

H-Store Architecture



H-Store Cluster

- Cluster = multiple computers (nodes)
- Node = multi-core CPUs, RAM
  - hosts multiple sites
- Site = process of H-Store
  - dedicated CPU core and RAM, data partition



Transaction Classes

1. Single-sited transactions
2. One-shot transactions

3. Two-phase transactions
4. Sterile transactions
5. General transactions

1. Single-sited transactions

- All queries hit the same partition

- Constrained Tree Schemas
  - Root table can be horizontally hash-partitioned
  - Collocate corresponding shards of child tables
  - No communication between partitions

2. One-shot transactions

- No inter-query dependencies

- Execute in parallel without communication
  - Replicate read-only parts
  - Vertical partitioning
  - Can be decomposed into single-sited plans
  - Local decisions → No redo log required

3. Two-phase class

- Two-phase classes
  - Phase 1: Read-only operations
  - Phase 2: Updates cannot violate integrity
  - No undo log required

4. Sterile classes

- Sterile classes
  - Operate independently
  - Do not depend on results / state of other concurrent transactions
  - No concurrency control needed
    - i.e., no coordination among transactions is necessary.

5. General transactions

- Require coordination with other transactions
  - read/write shared data;
  - update data in more partitions

Concurrency Control

- Run sterile, single-sited and one-shot transactions with no controls
- Other transactions with basic strategy
  - can escalate to intermediate or advanced
- Timestamp ordering of all transactions

Concurrency Control

- Basic Strategy
  - Coordinator sends tx subplans to “workers”
  - Worker waits for “small period of time”
    - to preserve timestamp order (network delay).
  - Worker executes the subplan
    - if there is not any uncommited, conflicting transaction
    - otherwise aborts.
  - Coordinators wait for “ok” from all sites and commits.

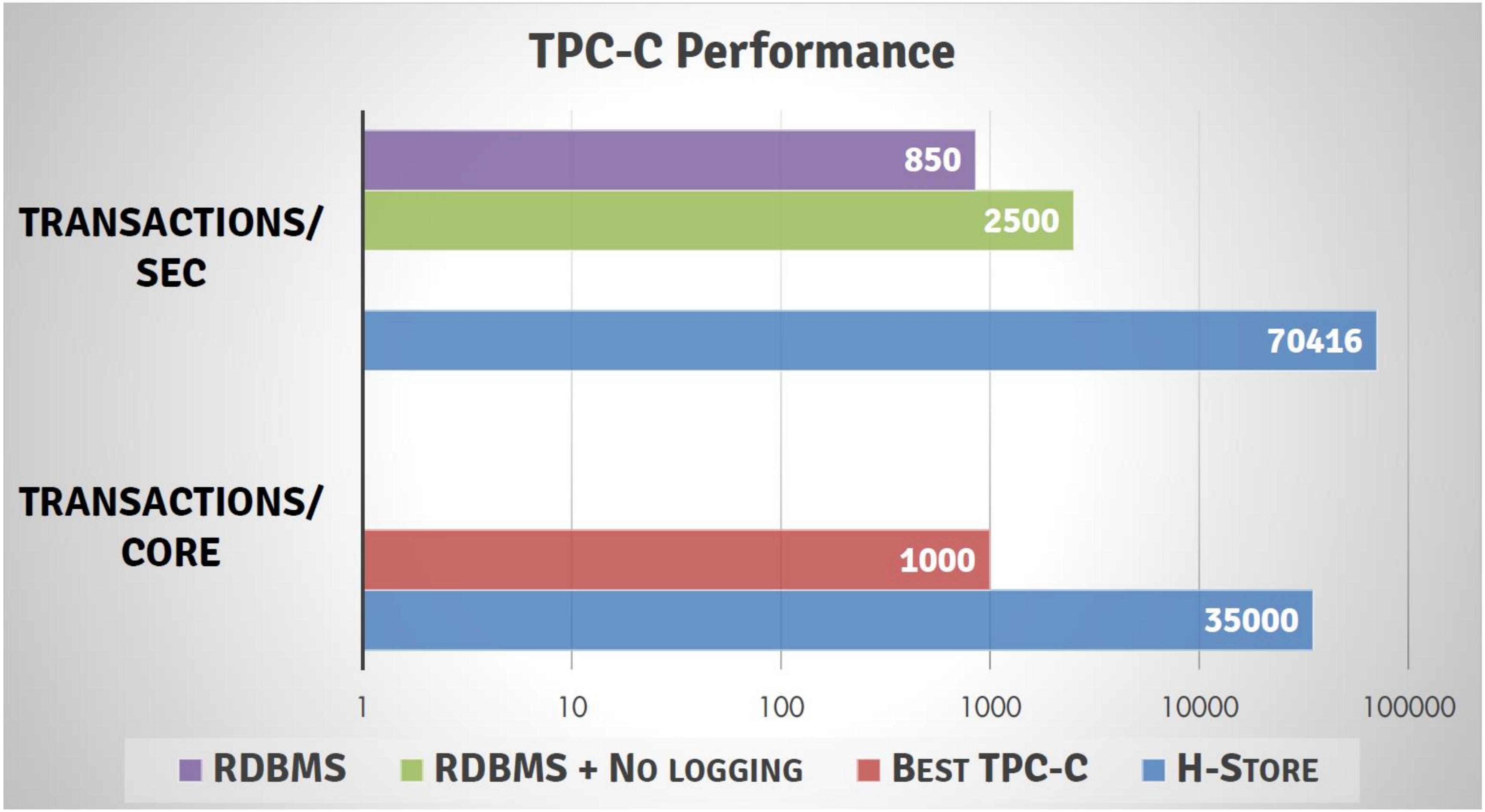
Concurrency Control

- Intermediate Strategy
  - if there are too many aborts with basic one
  - Increase wait latency in workers
- Advanced Strategy
  - if there are too many aborts with intermediate
  - == Optimistic concurrency control
  - Tracks read and write sets of each tx on each site
    - Aborts if a conflict between write and write is detected.

Database Layout

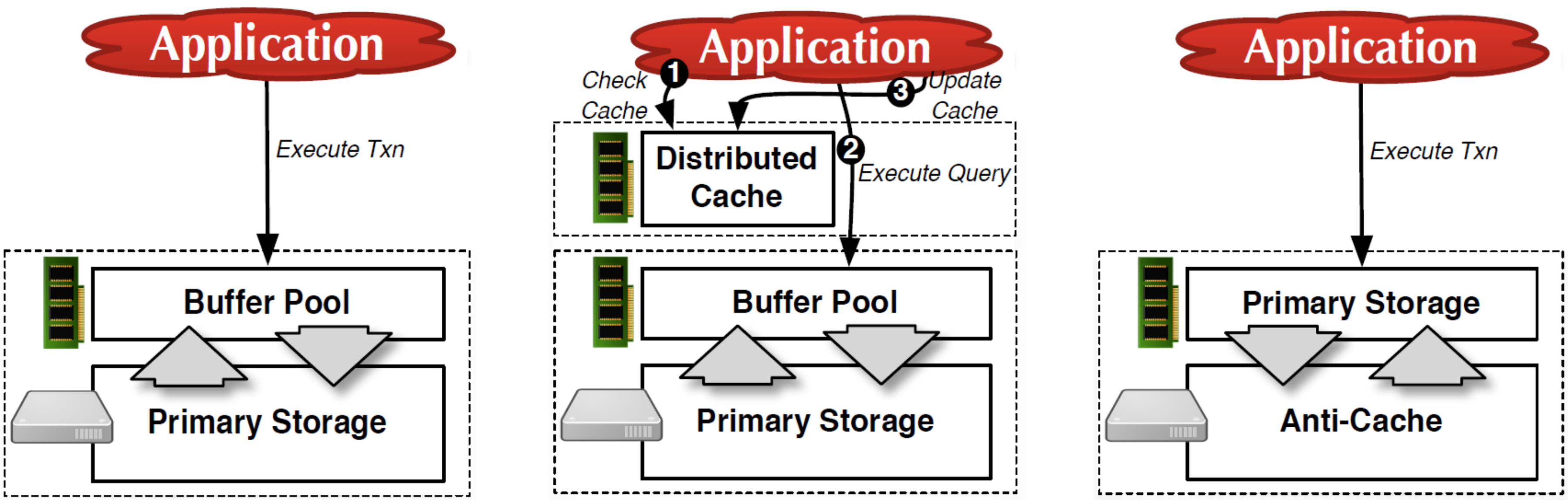
- Table replication
  - Read-only tables are on all sites
  - i.e., no communication → no latency
- Data partitioning
  - Horizontal partitioning into 4 partitions and 2 replicas
  - i.e., allow transaction execution in parallel
- K-safety of 2
  - Not enough RAM to replicate all tables
  - Every site is given a unique set of three partitions per table, thus preventing any pair of two sites from holding the only copies of a partition.

Performance comparison



Anti-caching (Durability)

- No logging is performed
- Cold data moved from RAM to disk
  - In a transactional-safe way



(a) Disk-oriented DBMS

(b) Disk-oriented DBMS with a Distributed Cache

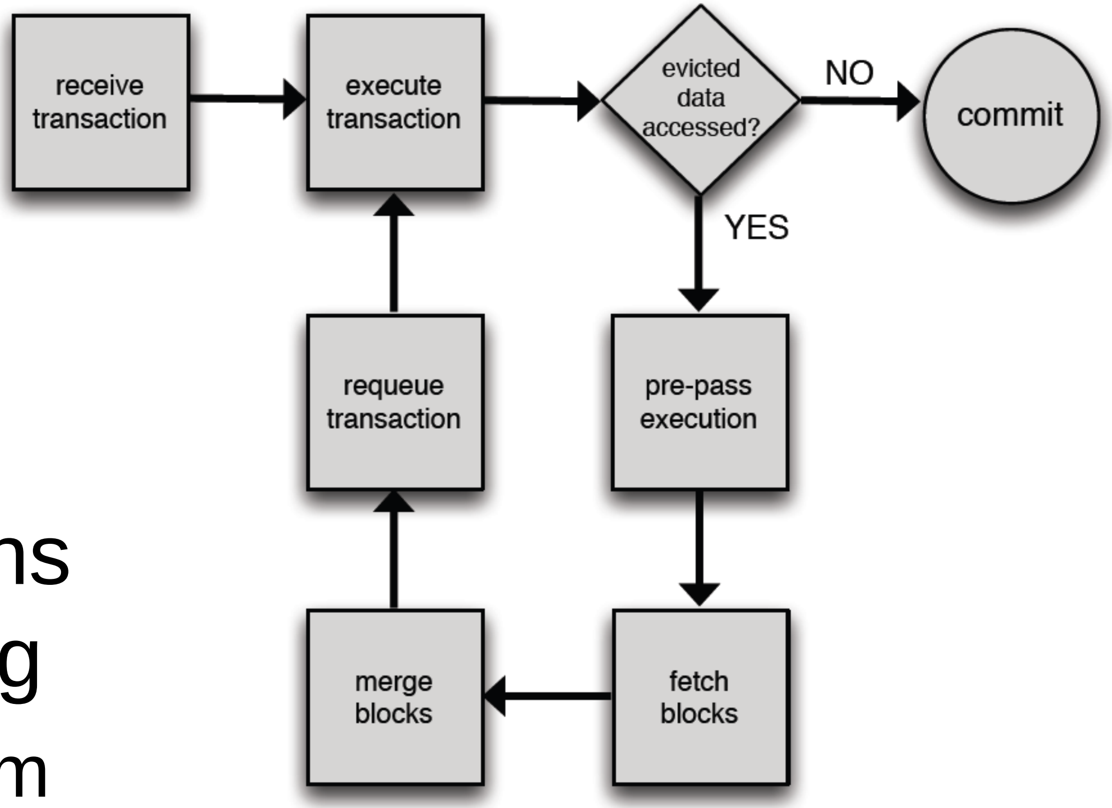
(c) Main Memory DBMS with Anti-Caching

Anti-caching

- Fine-grained eviction
  - Like in virtual memory mgmt, pages are copied.
  - Cold pages are written out.
  - A single hot tuple marks the page (block) hot.

- Non-blocking fetches
  - Abort transactions instead of waiting
    - for an I/O operation

Anti-caching



- Non-blocking fetches:
- Abort transactions instead of waiting
  - Reschedule them
  - Occurs if a transaction needs to operate on a tuple on disk
  - "pre-pass" tx to identify all evicted blocks.

H-store implementations

- Volt Active Data (VoltDB)
  - ensure “five 9’s” uptime
- SAP HANA
- SingleStore (MemSQL)
- eXtremeDB

Lecture Takeaways

- Trends in DBMS with current HW
- Main bottlenecks in full ACID systems
- NewSQL as H-Store
  - principle
  - transaction classes
  - durability