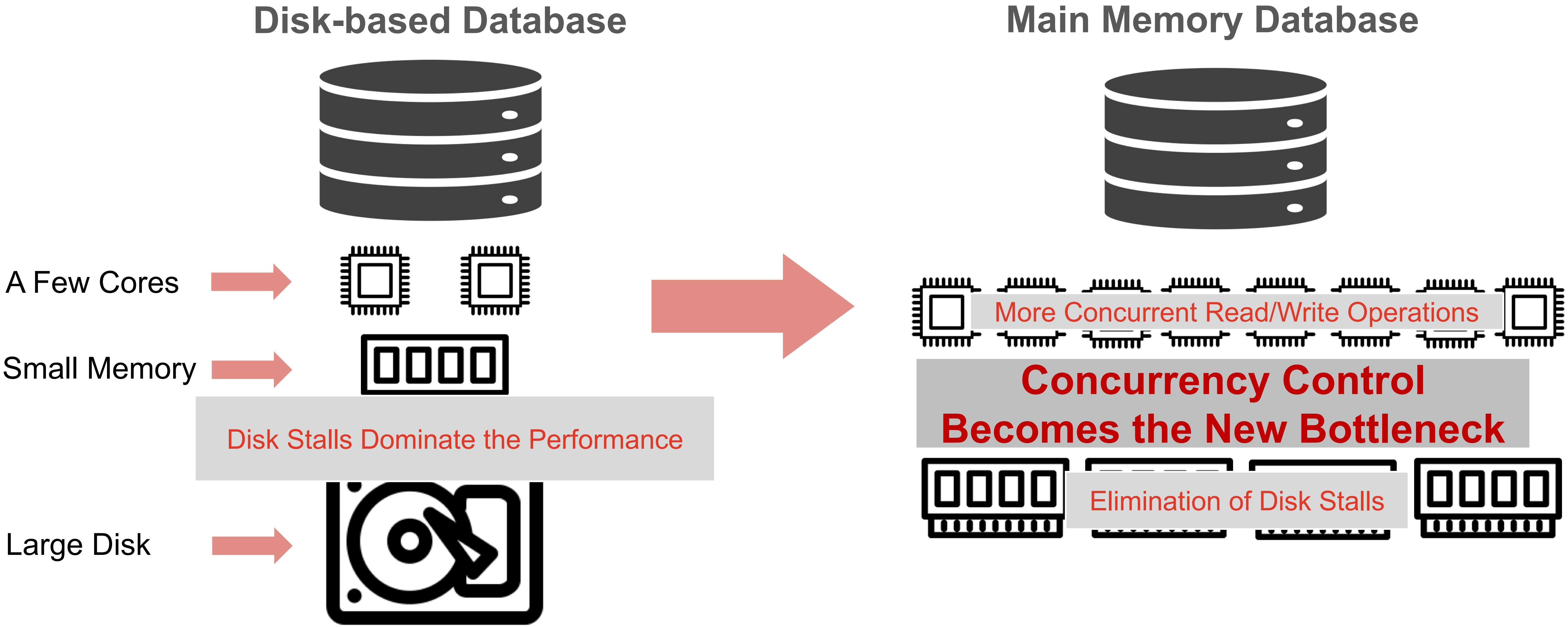
Toward Coordination-free and Reconfigurable Mixed Concurrency Control

Dixin Tang

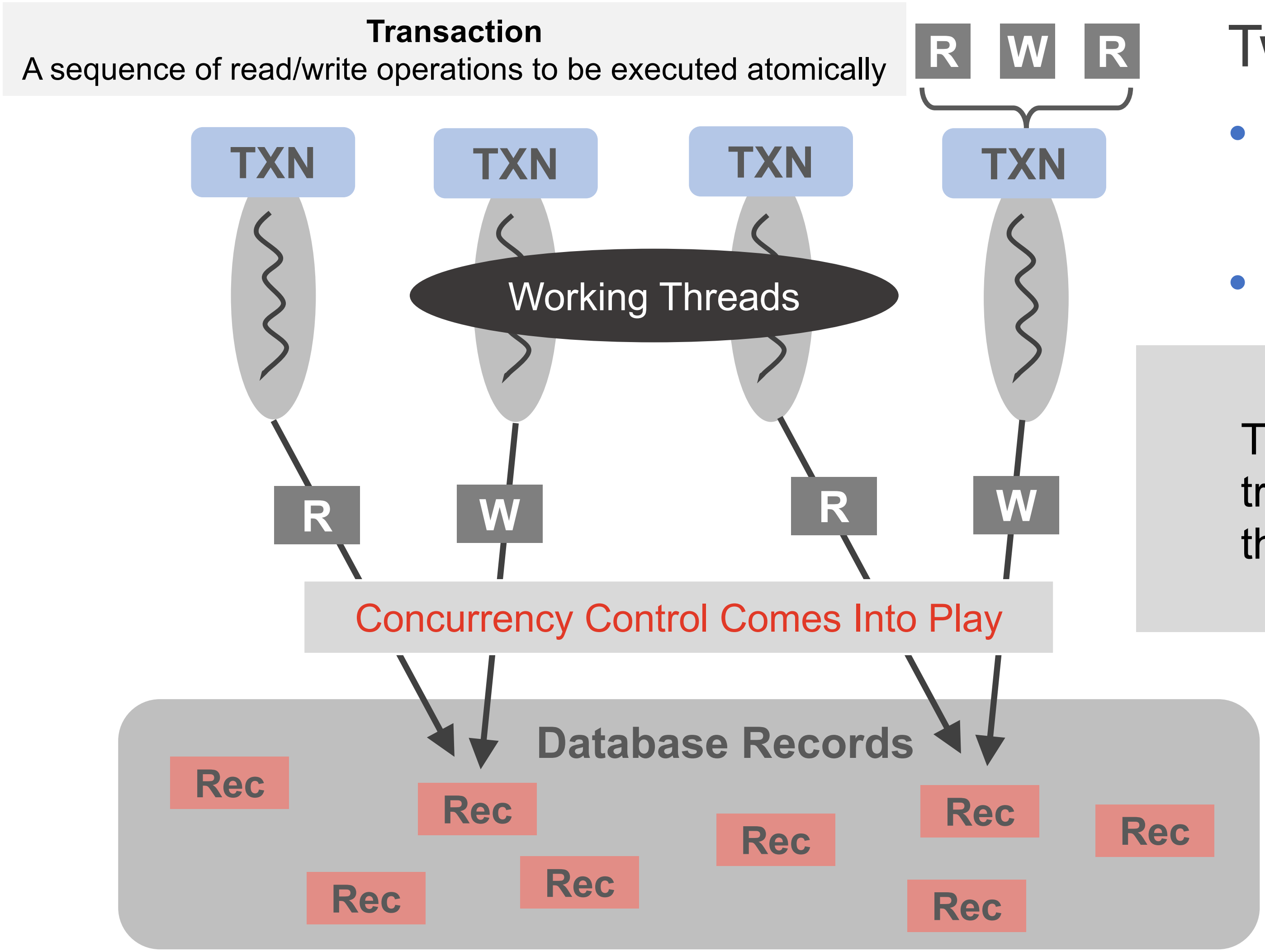
Aaron J. Elmore

THE UNIVERSITY OF CHICAGO

Hardware Development Changes Database Architecture



A Closer Look at Concurrency Control



Two Goals of Concurrency Control

- Interleaving concurrent operations to maximize the performance
- Guarantee consistency

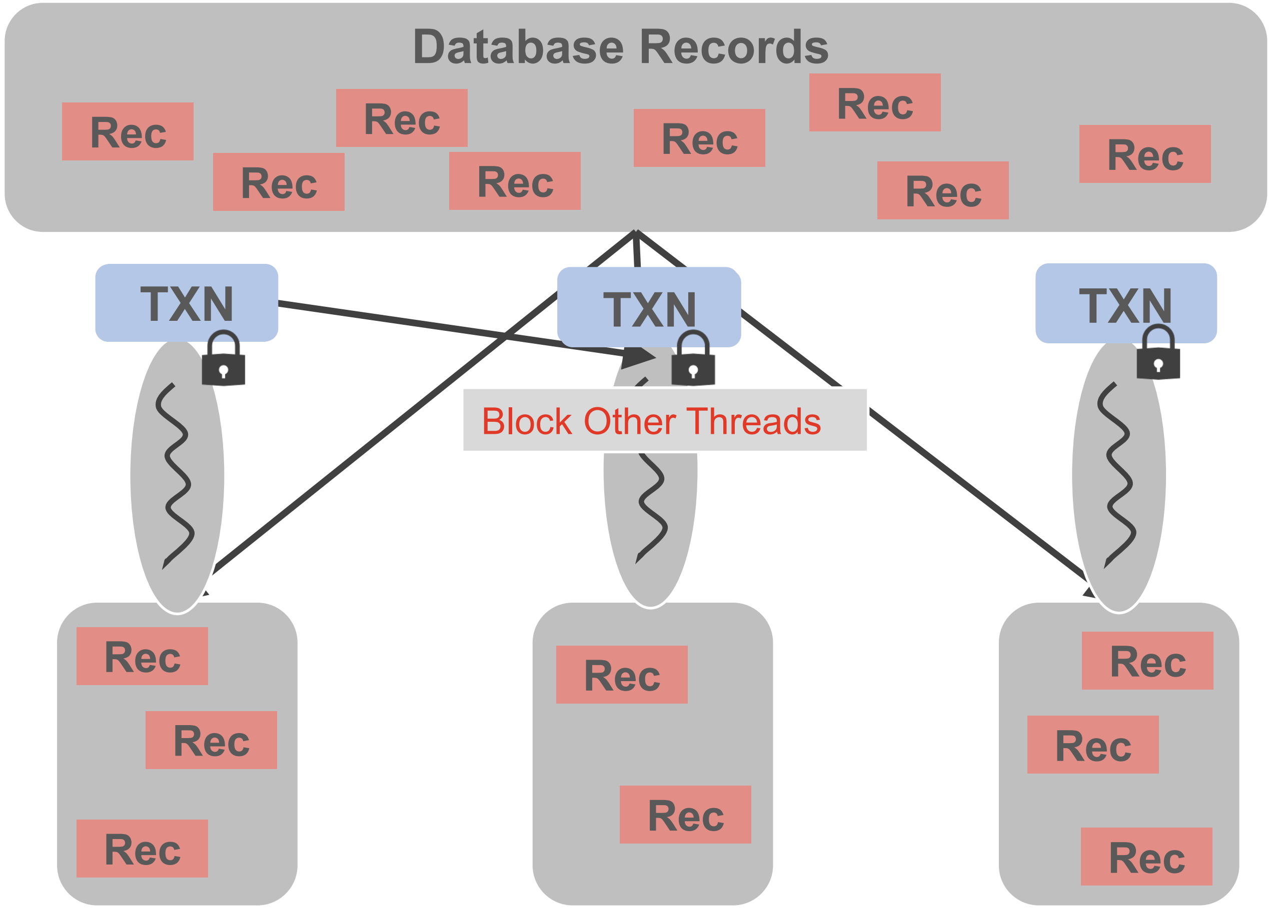
Serializability

The result of interleaved execution of concurrent transactions is equivalent to the result of executing these transactions in one serial order

One Concurrency Control Does Not Fit All

PartCC (Partition-based single-thread concurrency control)

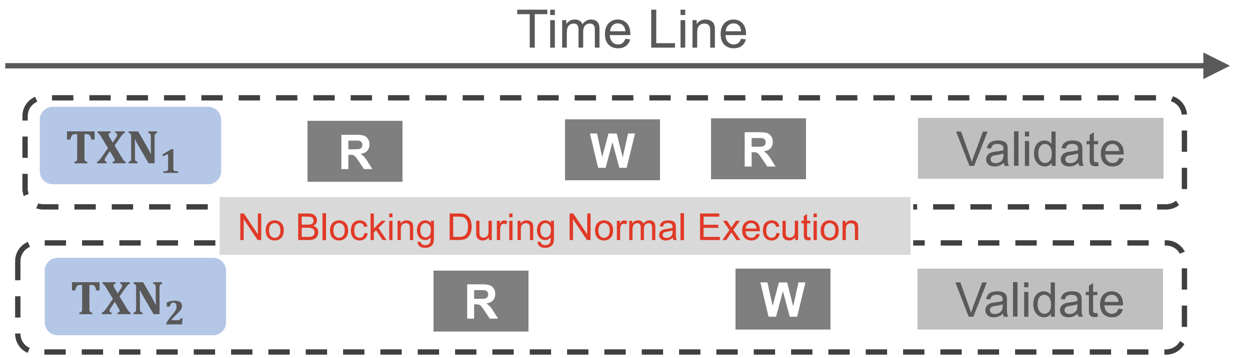
- Perform well under partitionable workloads
- Cross-partition transactions hurt the performance



One Concurrency Control Does not Fit All

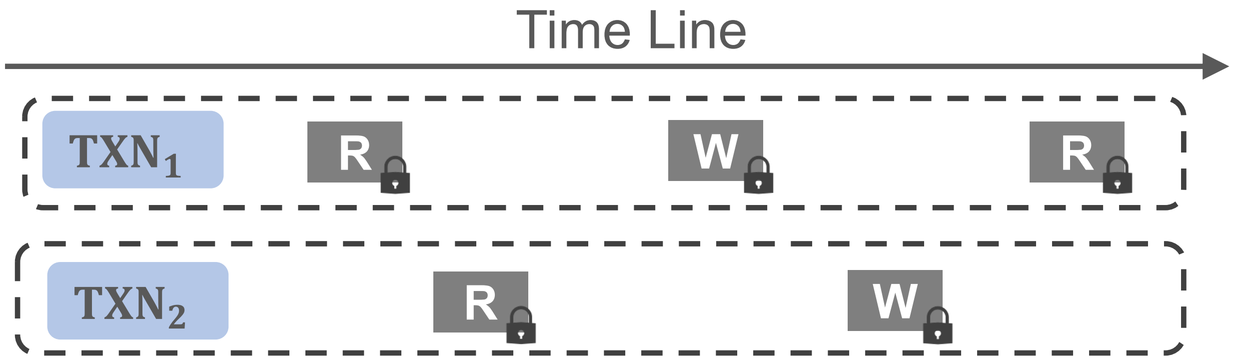
OCC (Optimistic Concurrency Control)

- Perform better under low-conflict workloads
- Conflicts can hurt the performance because transactions need to be restarted if validation fails



2PL (Two Phase Locking)

- Perform better under highly-conflicted workloads
- Concurrency control overhead and synchronization overhead



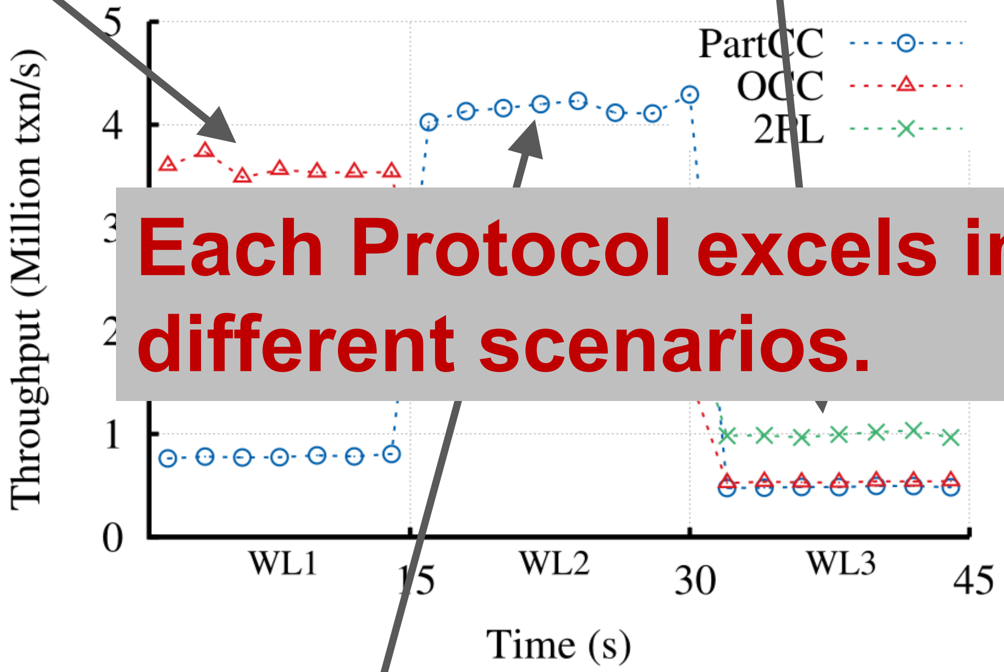
One Concurrency Control Does not Fit All

experiment on our main-memory database prototype using YCSB workloads

- OCC from Silo [1]
- 2PL using VLL [2]
- PartCC from H-Store [3]

Not partitionable, Read Only

Not partitionable, High skew, Write heavy

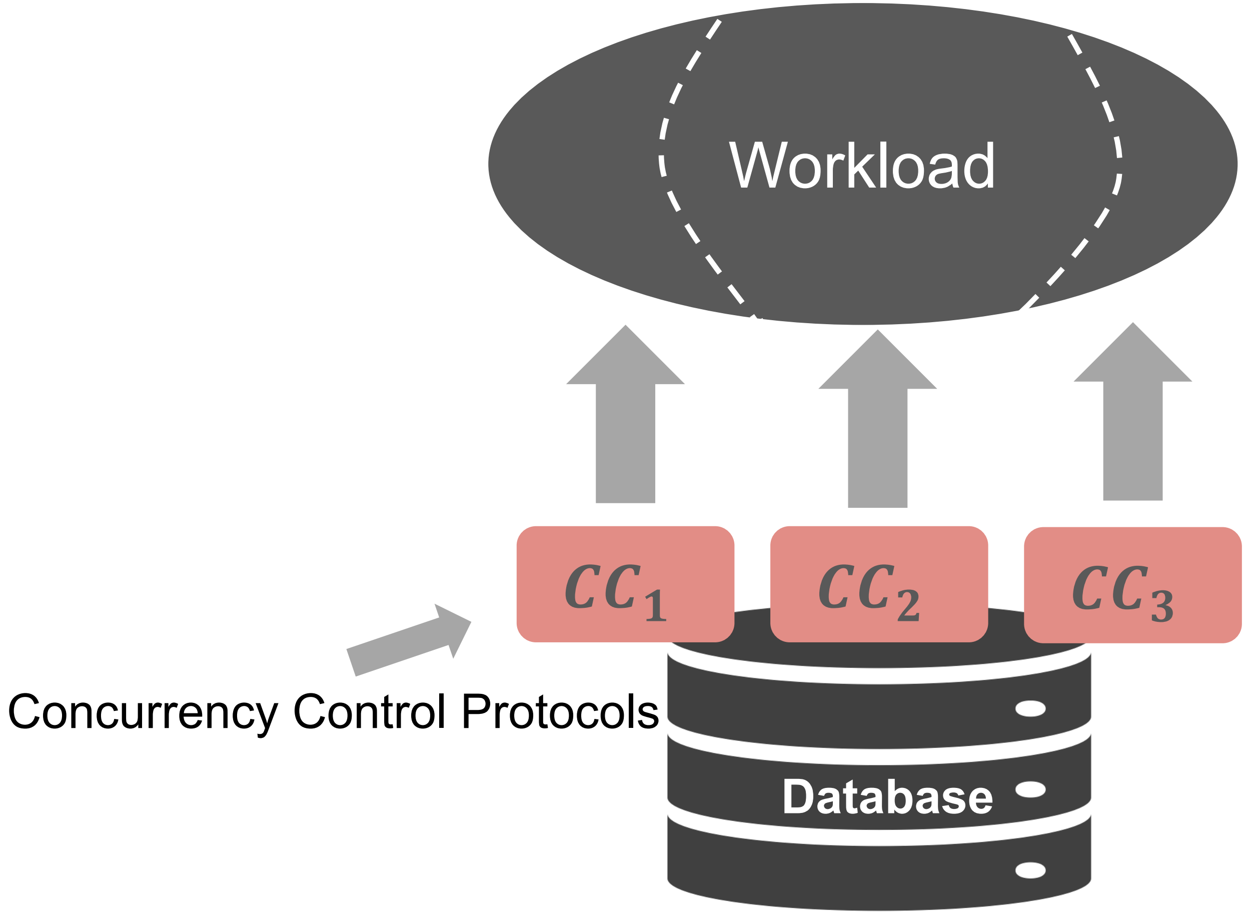


Each Protocol excels in different scenarios.

Partitionable

[1] TU, S., et al. Speedy transactions in multicore in-memory databases. SOSP' 13
[2] REN, K. et al. Lightweight locking for main memory database systems. PVLDB'12
[3] KALLMAN, R., et al. H-store: a high-performance, distributed main mem-ory transaction processing system. PVLDB'08

A General Solution: Mixed Concurrency Control



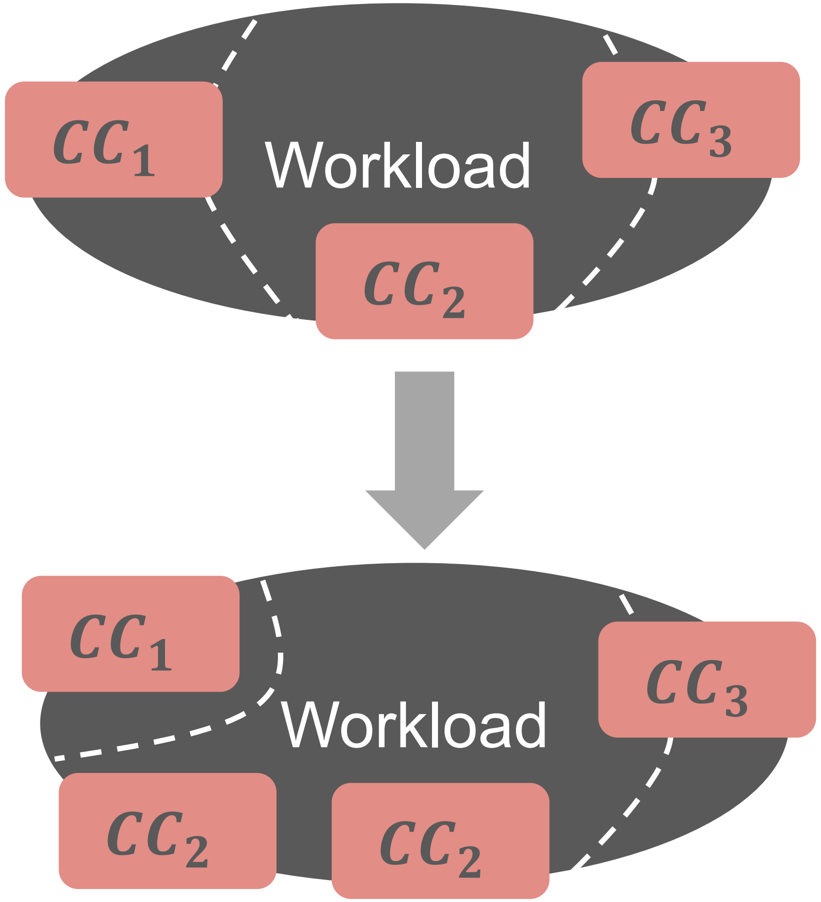
Two Benefits

- Each protocol can process the part of workload it is optimized for
- Each protocol can avoid being brittle to workload where it does not perform well

Two Challenges of Mixed Concurrency Control

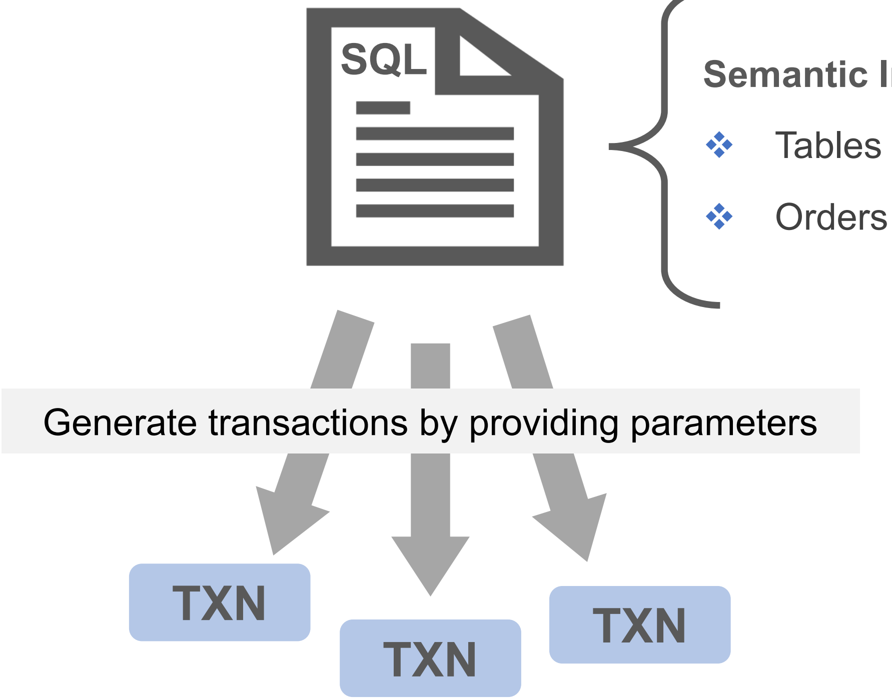
- How to partition a workload and mix multiple concurrency control protocols efficiently

- How to reconfigure a protocol when the workload changes



Previous Approach of Mixed Concurrency Control: Partition Stored Procedures by Conflicts

Stored Procedure (SP)
A Parameterized Transaction Template



Semantic Information

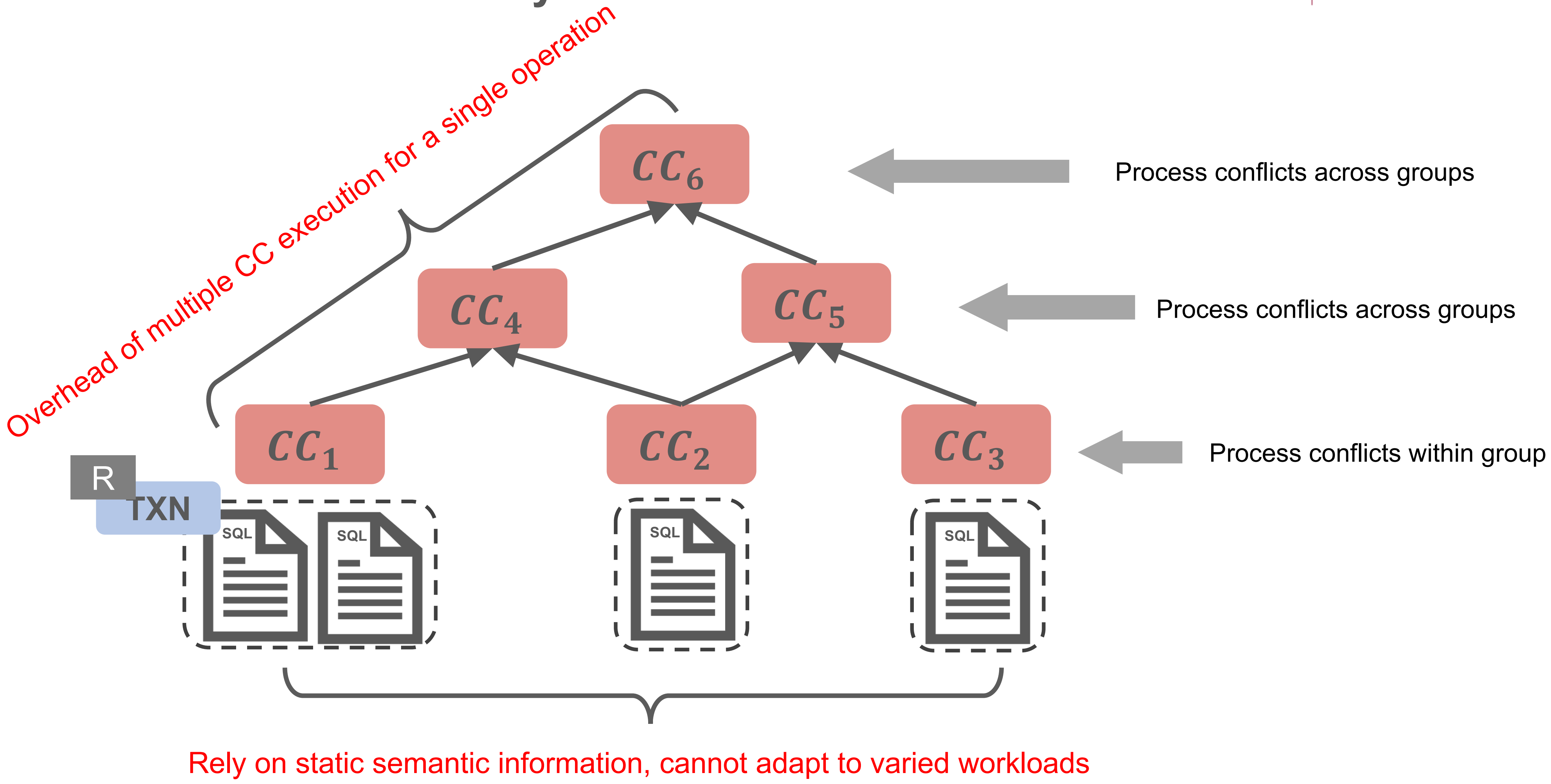
- Tables to be read/written
- Orders of reading/writing tables

Previous Approach [1]

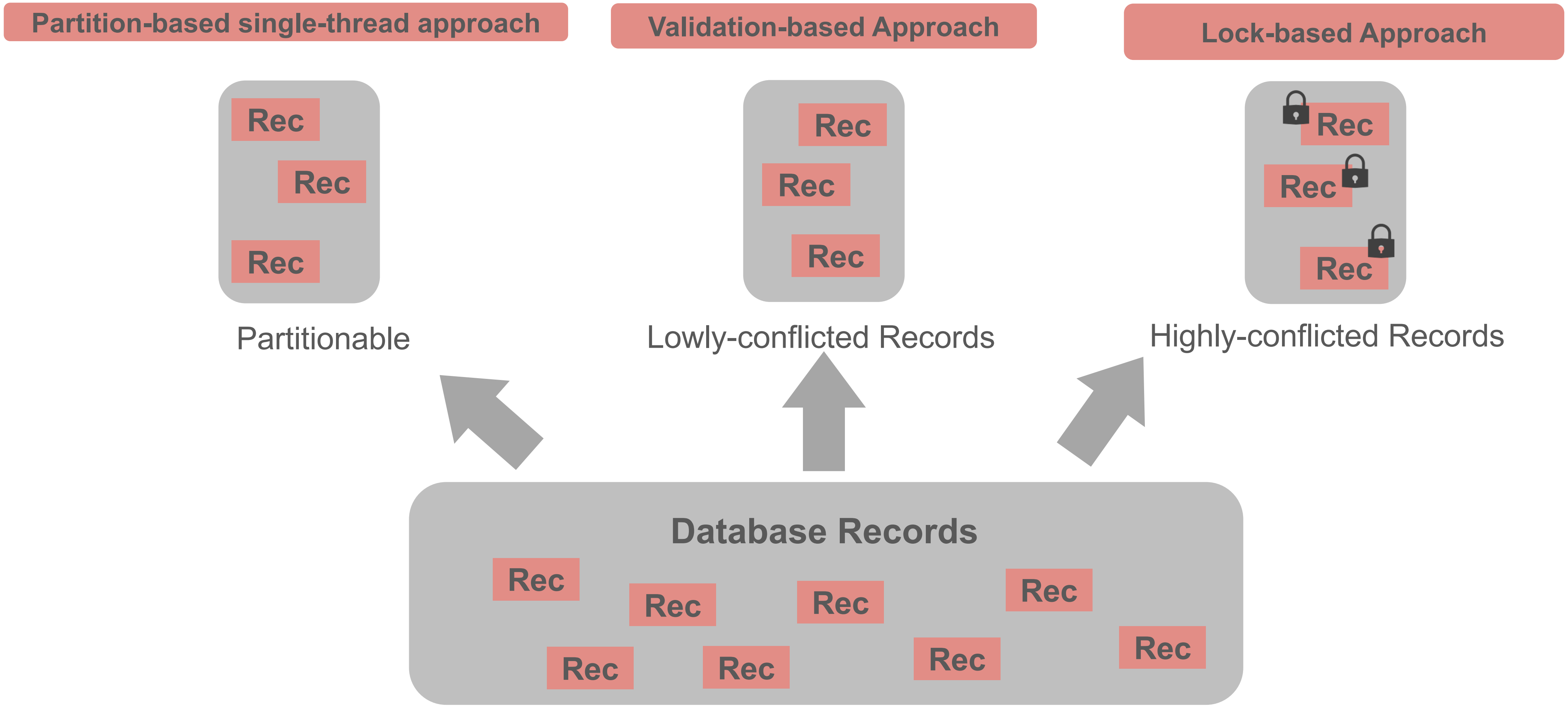
- Group stored procedures by conflicts extracted from their semantic information
- Assign each group a protocol to process conflicts within that group
- Need additional concurrency control protocols to process conflicts across groups

[1] SU, C., et al. Bringing modular concurrency control to the next level. SIGMOD' 17

Previous Approach of Mixed Concurrency Control: Partition Stored Procedures by Conflicts

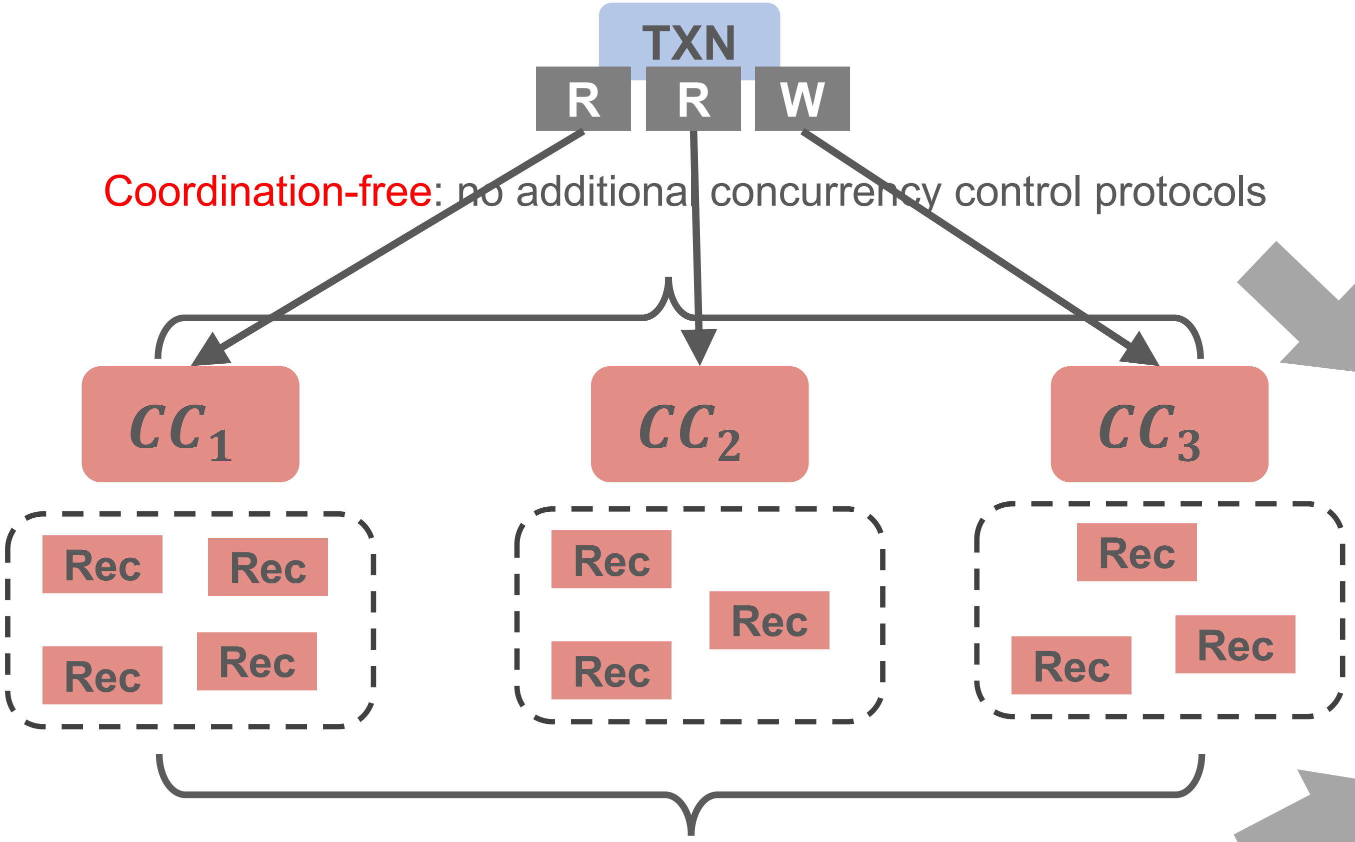


A New Perspective: Partition Records by Access Characteristics



CormCC: Coordination-free and Reconfigurable Mixed Concurrency Control

THE UNIVERSITY OF CHICAGO



Reconfigurable: partitioning records depend on real-time data access characteristics, make online protocol reconfiguration possible

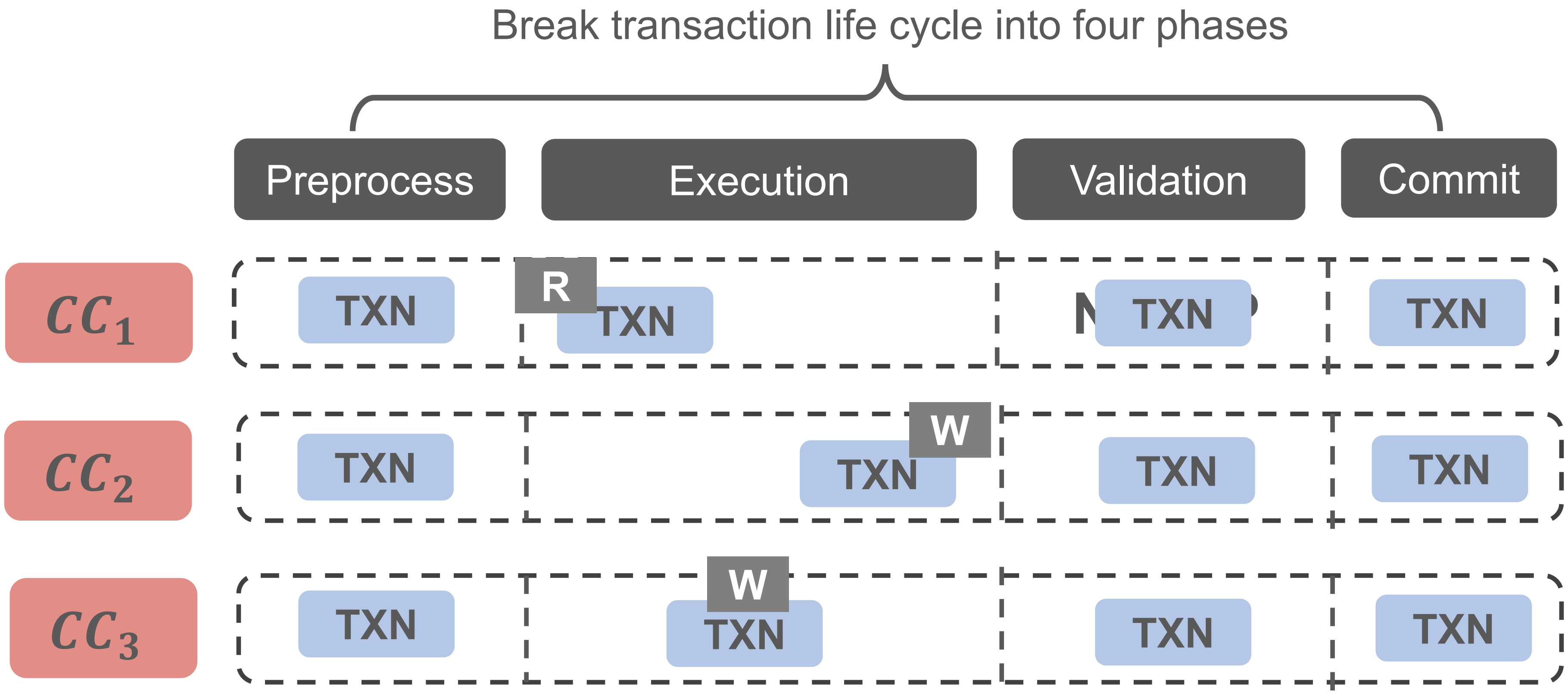
Our Approach

- Partition database records and assign each partition a single protocol
- A single protocol is used to process all operations for that partition of records

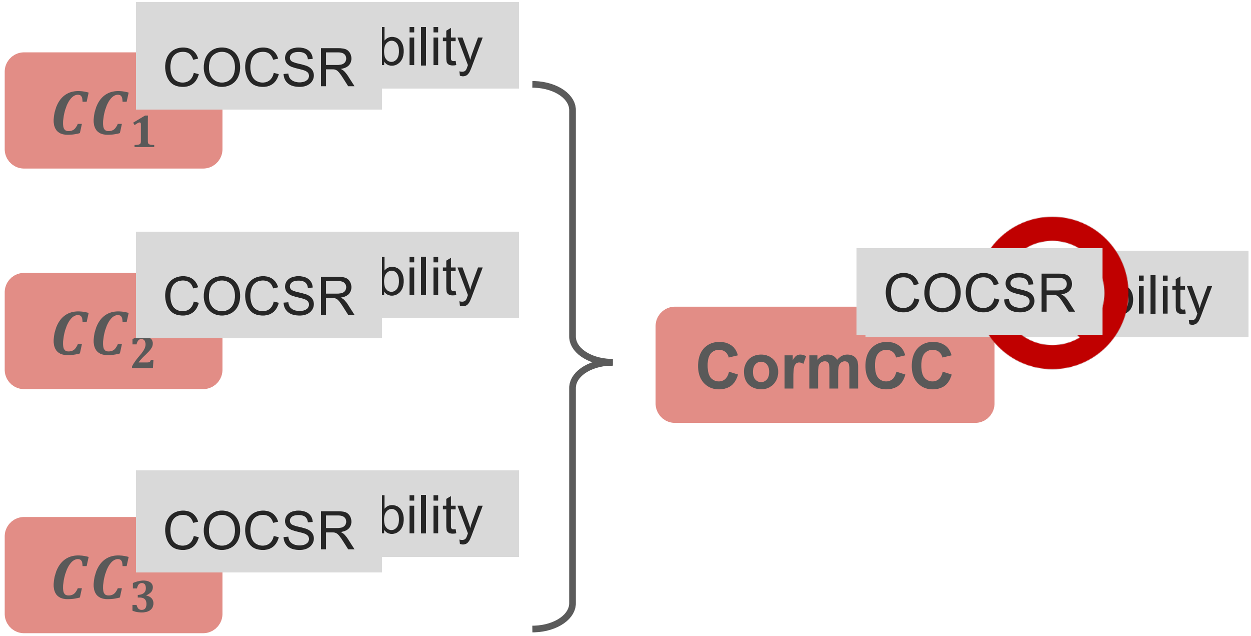
To achieve the two goals, we need to answer four questions:

- **How does CormCC execute**
- **How to maintain serializability**
- **How to guarantee deadlock free**
- **How to enable online protocol switch**

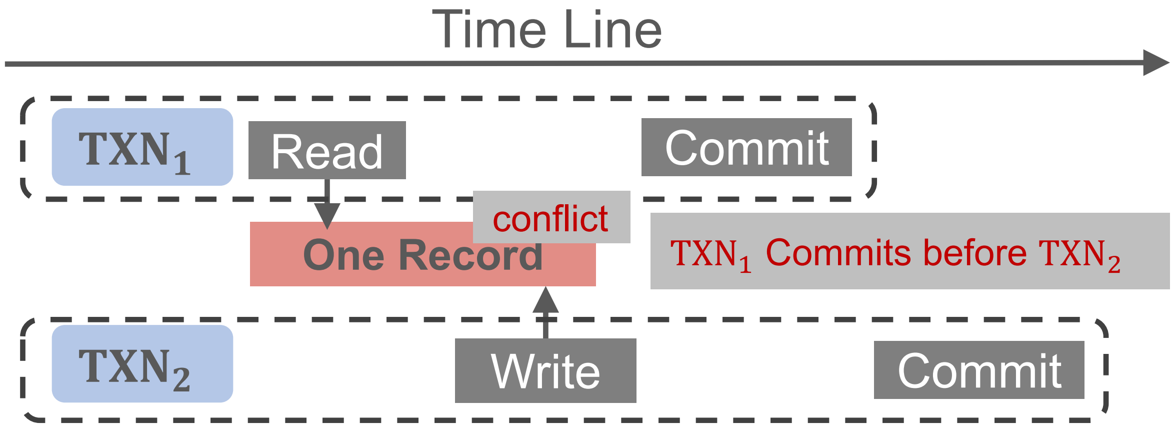
CormCC Execution



Correctness of CormCC – Serializability

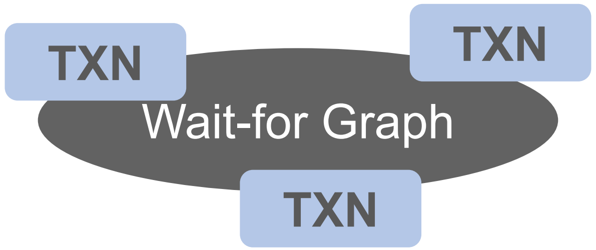


- COCSR (Commit ordering conflict serializable)
  - Sufficient condition of serializable
  - Commit ordering respects conflicts
- If all protocols are COCSR, then CormCC is COCSR
  - Proof can be found in the paper



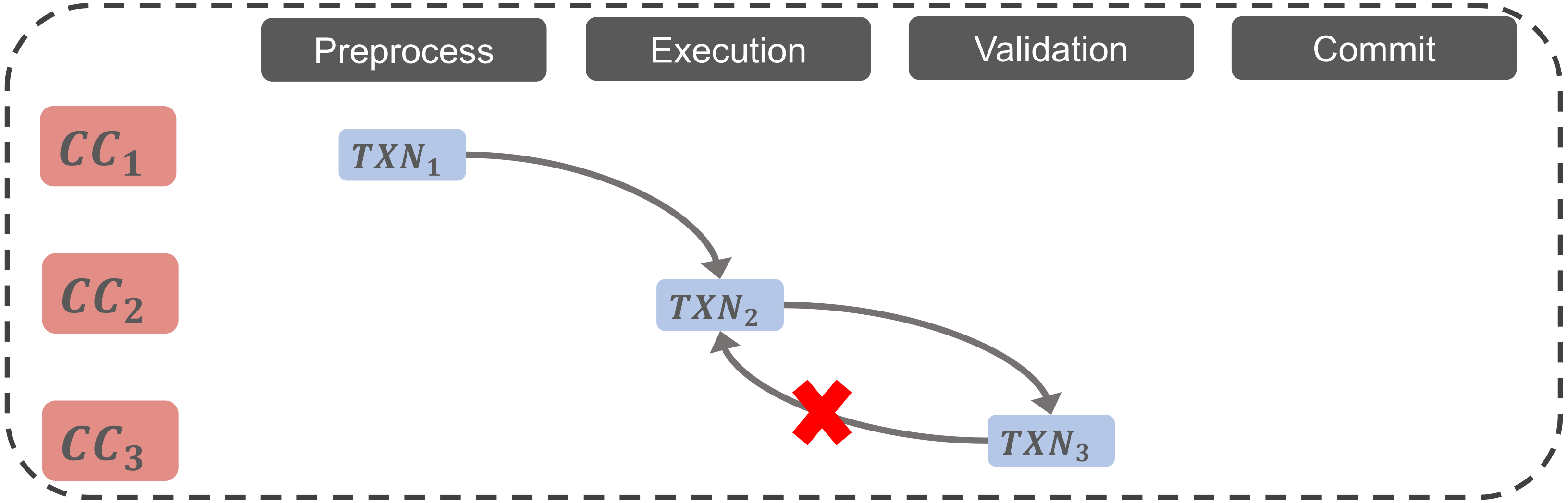
CormCC – Deadlock Free

Deadlock Detection



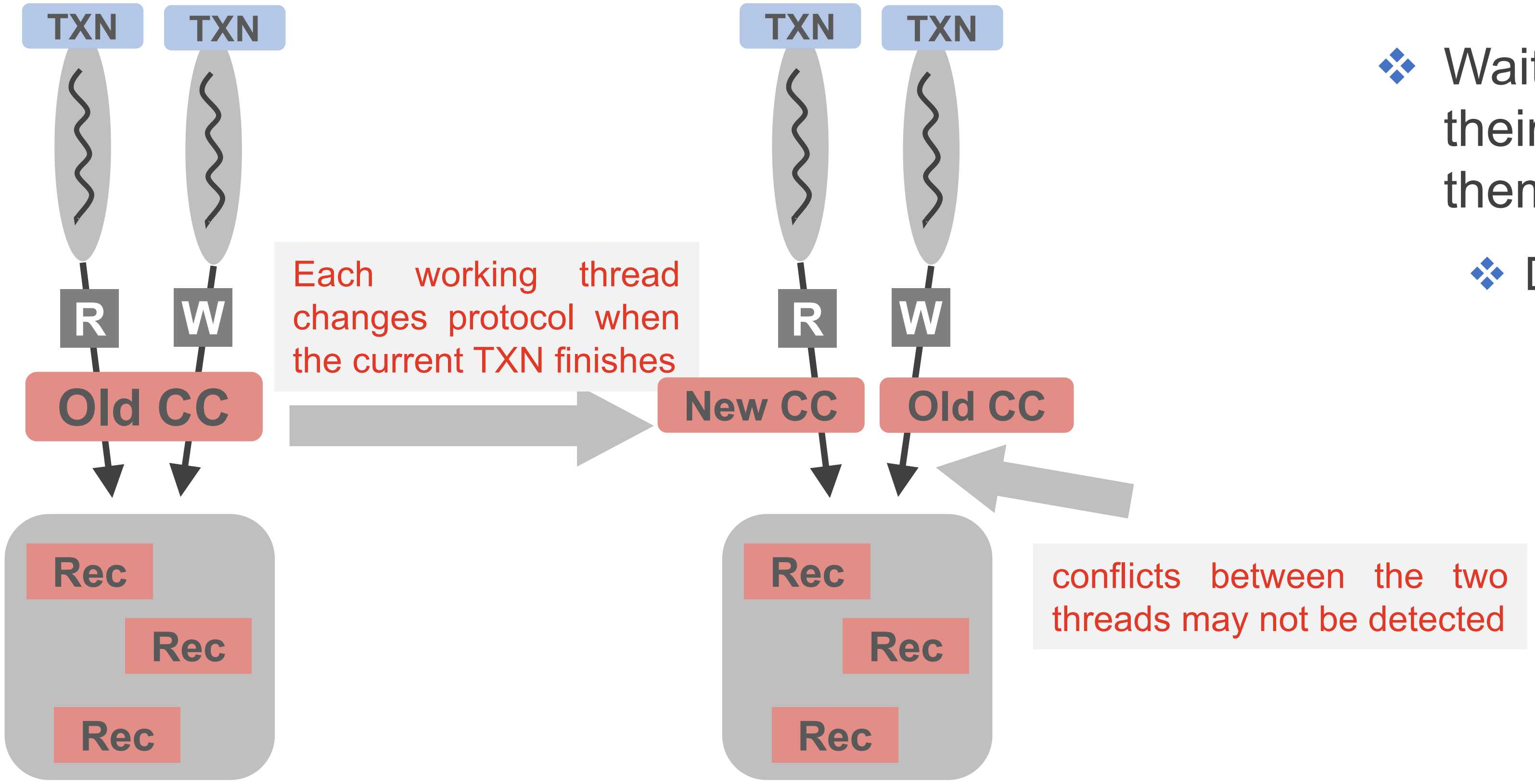
Our Approach: Deadlock Prevention

- We require each protocol can only exclusively let transactions wait in no more than one phase
  - No deadlock within one phase
  - Transactions in earlier phases can wait for later phases, but not the other way around



CormCC – Online Protocol Reconfiguration

Online reconfiguration can cause inconsistency

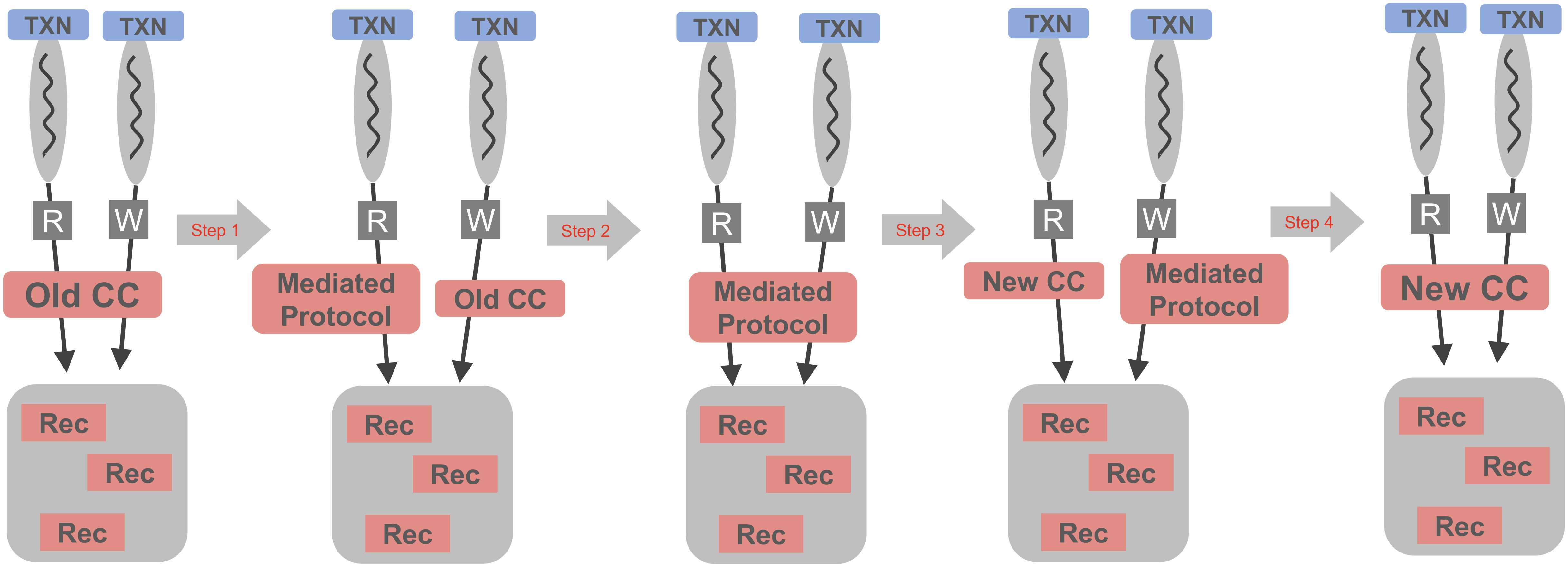


A straightforward solution: stop all

- Waiting all working threads to complete their current transactions, and stop them from receiving new transactions
  - Decrease the performance of database

CormCC – Online Protocol Reconfiguration

Our solution: using a mediated protocol that is compatible to both old and new protocols



CormCC – Online Protocol Reconfiguration

THE UNIVERSITY OF CHICAGO

How to Build a Mediated Protocol

- The mediated protocol executes the logics of both old and new protocol
- Example: Mediated Protocol between OCC and 2PL

Mediated Protocol between OCC and 2PL

Read

- Read timestamp (OCC)
- Apply read lock (2PL)

Write

- Write to a local buffer (OCC)
- Apply write lock (2PL)

Validate

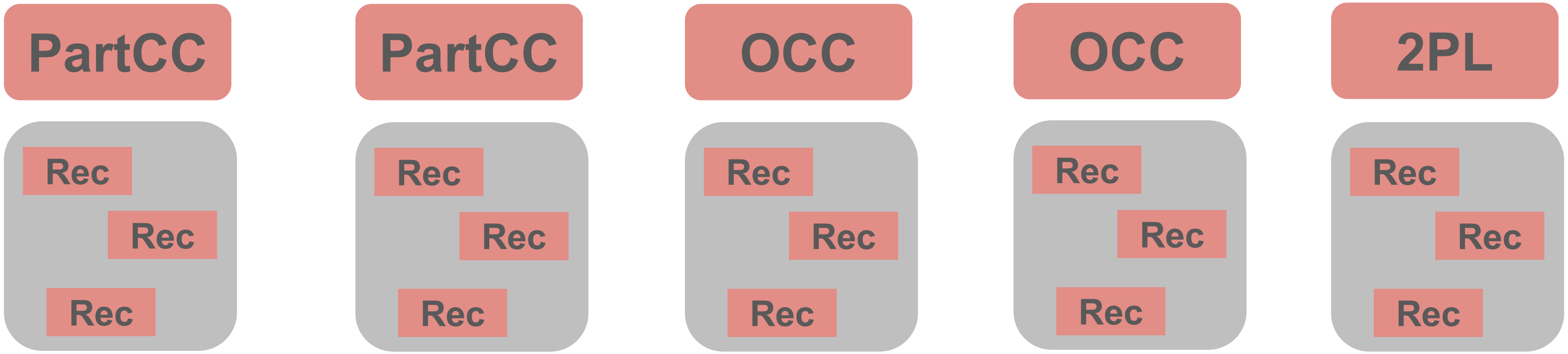
- Execute Validate phase of OCC

Prototype Design

THE UNIVERSITY OF CHICAGO

- Supporting PartCC from H-Store [3], OCC from Silo [1], and 2PL from VLL [2]

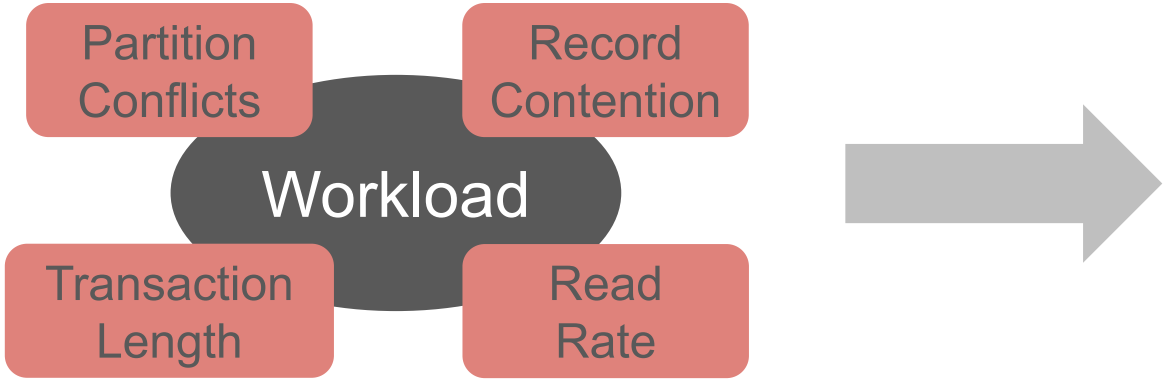
- Partition the whole database and apply each partition a single protocol

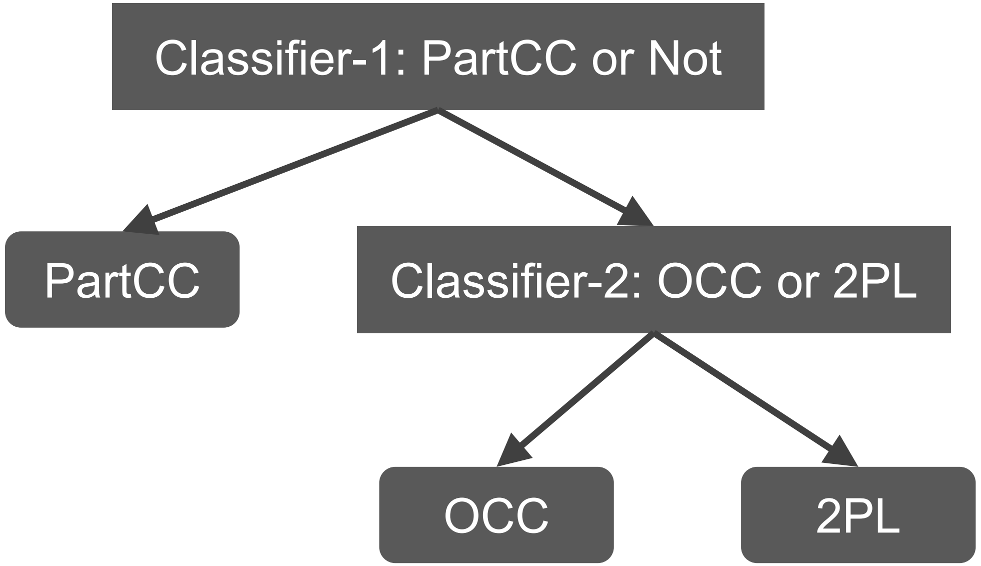


[1] TU, S., et al. Speedy transactions in multicore in-memory databases. SOSP' 13
[2] REN, K. et al. Lightweight locking for main memory database systems. PVLDB'12
[3] KALLMAN, R., et al. H-store: a high-performance, distributed main mem-ory transaction processing system. PVLDB'08

Prototype Design

- Selecting the ideal protocol for each partition
  - Feature Engineering: design several features to capture the performance difference of candidate protocols
  - Classifiers: building a two-layer classifier





Experiments

THE UNIVERSITY OF CHICAGO

Experiment Settings

❖ A Machine with 32 cores, 256 GB main memory

Workload

❖ TPC-C: Order processing application with 5 stored procedures, 32 warehouses

❖ YCSB: One Table with 1 million records, each with 25 columns and 20 bytes for each column; One type of transaction mixed with read or read-and-modify operations

❖ Partition into 32 partitions (i.e. equal to the number of cores)

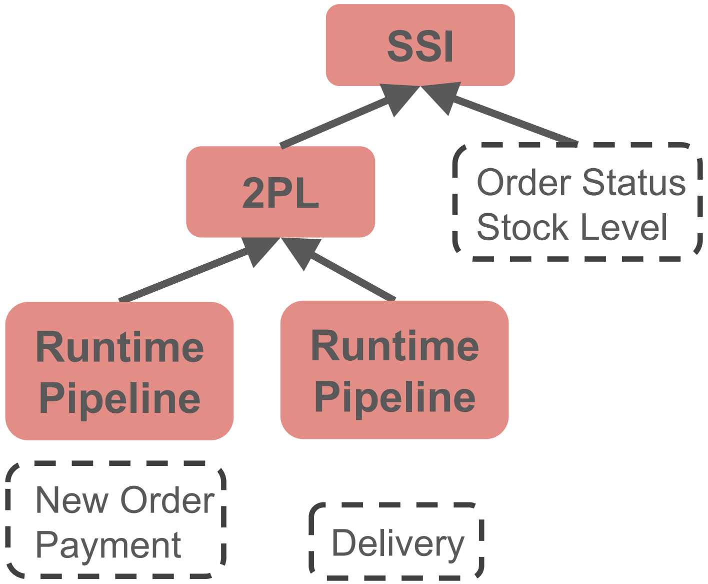
Experiments

THE UNIVERSITY OF CHICAGO

Our experiments compare CormCC with

- Single candidate protocols
- Hybrid - Hybrid OCC and 2PL execution based on MOCC [1] /Hsync [2]
- Tebaldi - A stored procedure-oriented general framework of mixed concurrency control [3]

Tebaldi Configuration for TPC-C (from Tebaldi paper)

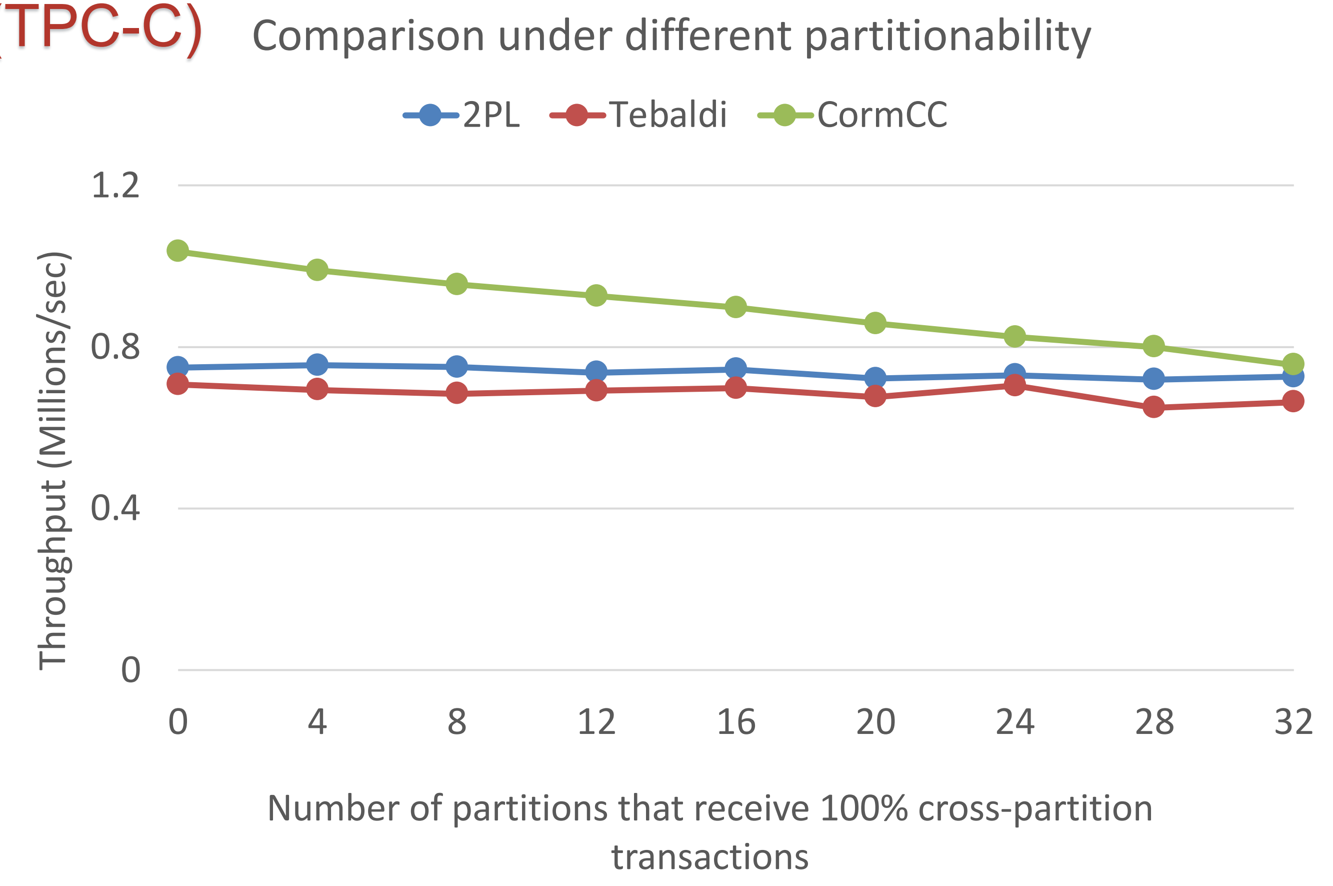


[1] WANG, T., et al. Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. PVLDB'16
[2] SHANG, Z., et al. Graph analytics through fine-grained parallelism. SIGMOD'16
[3] SU, C., et al. Bringing modular concurrency control to the next level. SIGMOD' 17

Experiments – Compare with Tebaldi

Comparison under different partitionability (TPC-C)

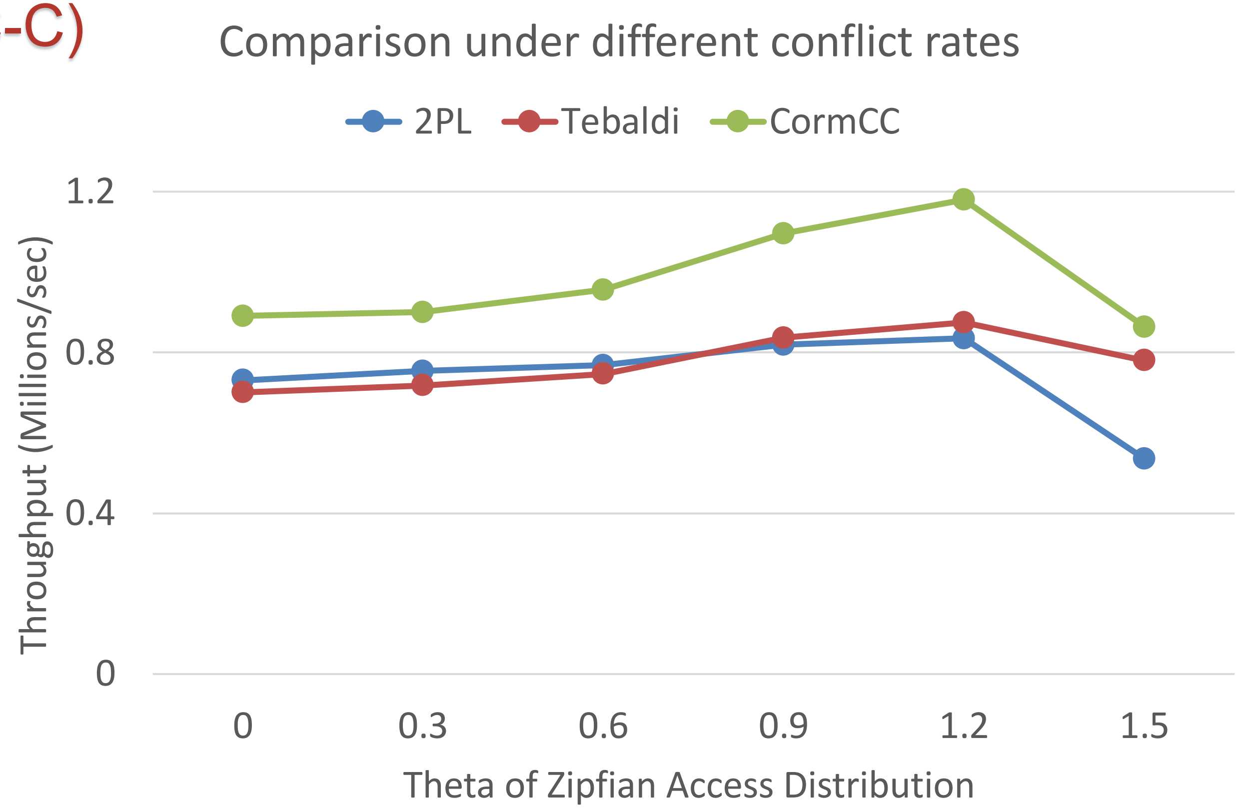
- Start with well-partitionable workload
  - Each partition receives 100% single-partition transactions
- Increase the number of partitions receiving 100% cross-partition transactions



Experiments – Compare with Tebaldi

Comparison under different conflict rates (TPC-C)

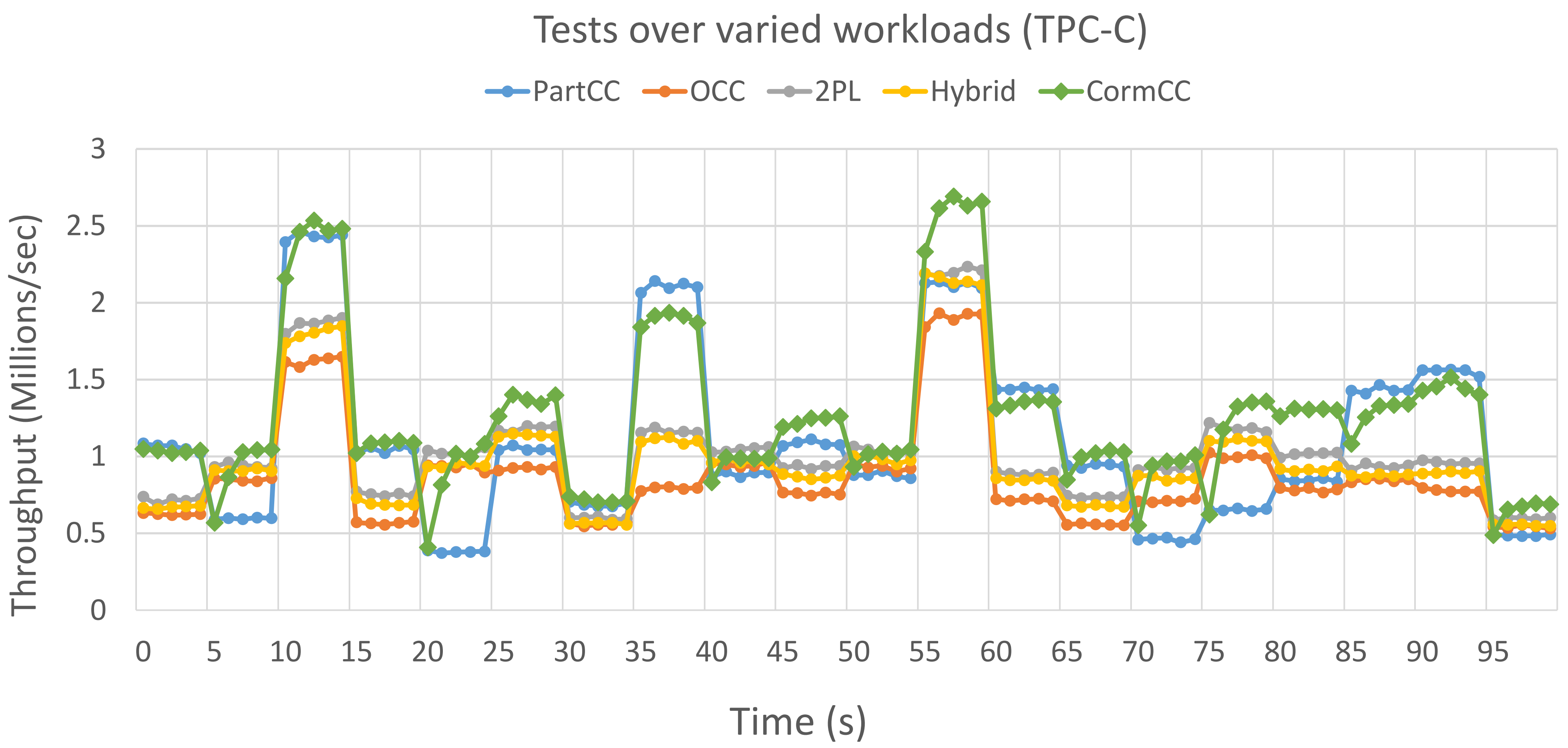
- We modify TPC-C to increase access skewness
  - We use Zipfian distribution and increase its theta parameter to introduce higher conflict rate



Experiments – Varied workloads

❖ Vary parameters every 5 seconds:

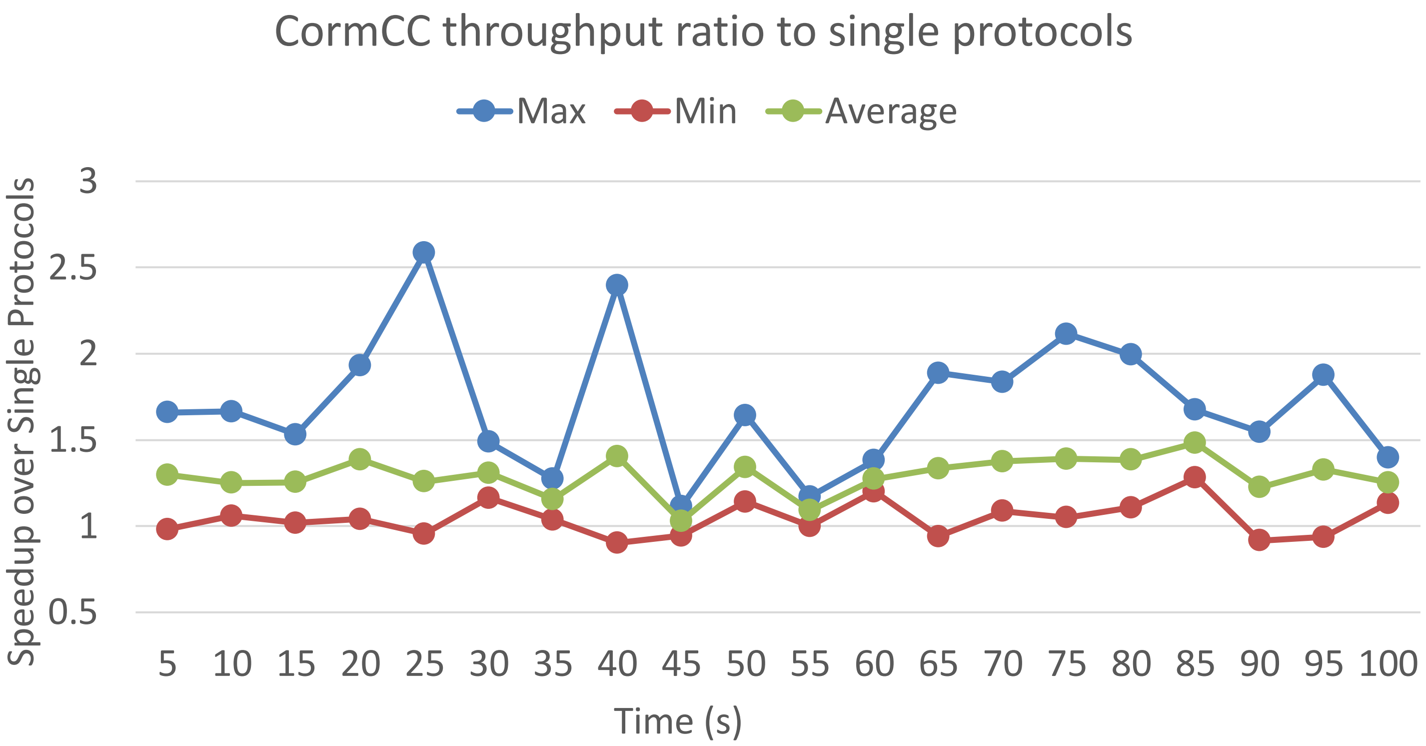
Transactions mix, Percentages of cross-partition transactions, Access skewness (i.e. theta of Zipf)



Experiments – A closer look at the same results of varied workload tests

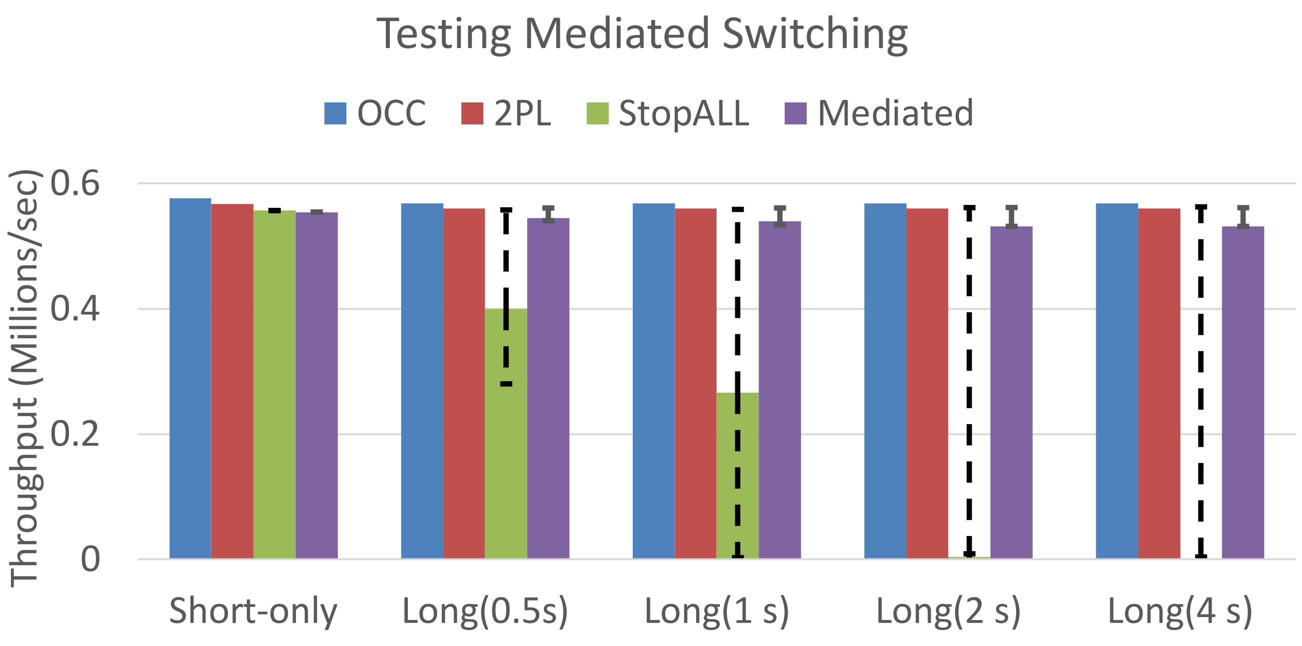
THE UNIVERSITY OF CHICAGO

- We aggregate the throughput of every 5s and report CormCC throughput ratio to single protocols
  - Max: Speedup over the worst single protocols
  - Min: Speedup over the best single protocols
  - Average: Speedup over average throughput of single protocols



Experiments – Mediated Protocol Switch

- Switch from OCC to 2PL using YCSB workloads
- Report the throughput during protocol switch
- Compared with StopALL
- Test a workload with short-only transactions and workloads with one long transaction of different duration
  - 0.5s, 1s. 2s, 4s
- Test different switching points
  - At the beginning of the long transaction
  - At the middle of the long transaction
  - At the end of the long transaction



Conclusion

THE UNIVERSITY OF CHICAGO

- CormCC, a general mixed concurrency control framework that does not introduce any coordination overhead and supports online reconfiguration

- Experiments show that CormCC can achieve significant throughput improvement over single static protocols and state-of-the-art mixed approaches.

Thanks!

Backup Slides

CormCC Execution in Prototype

	Preprocess	Execution	Validation	Commit
PartCC	✓ ✗	✓		✓
OCC		✓	✓ ✗	✓
2PL		✓ ✗		✓

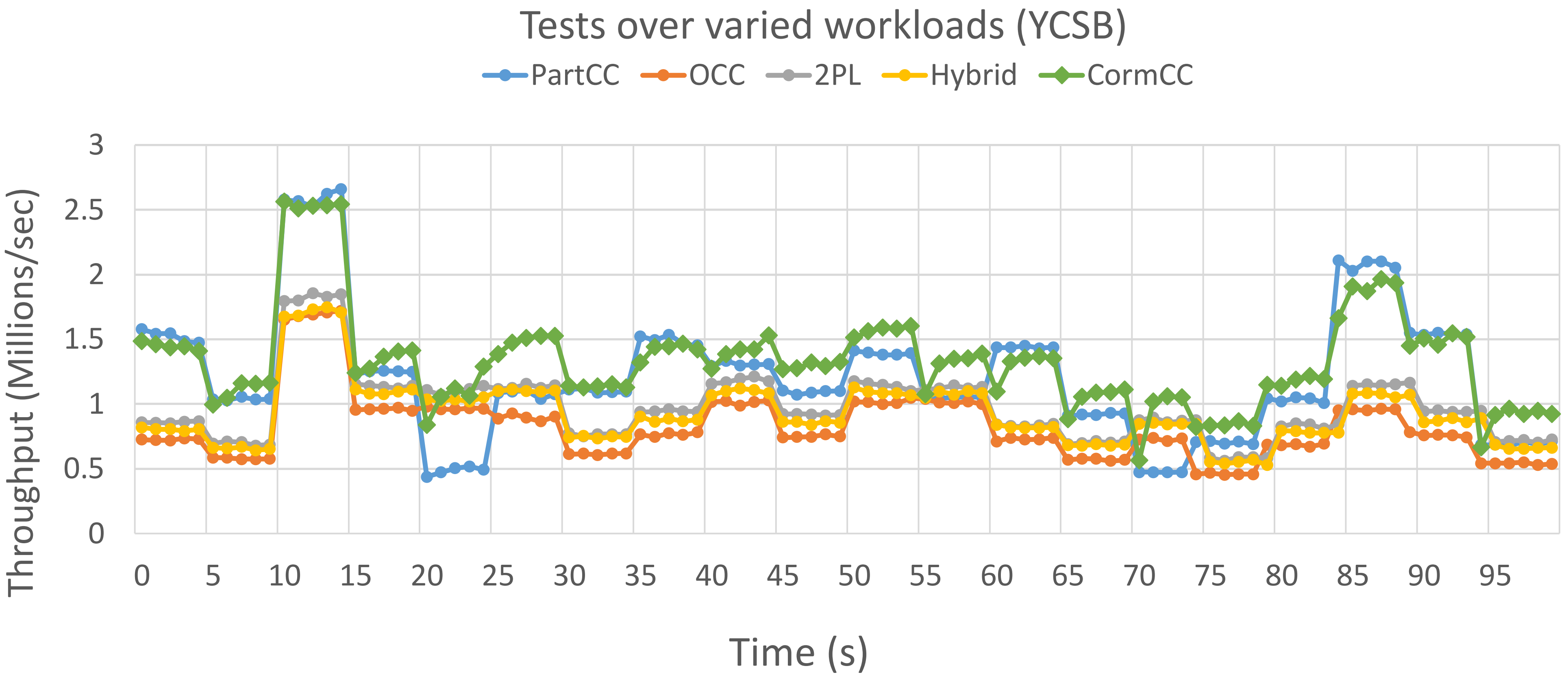
✓ This protocol includes this phase

✗ This protocol makes transaction wait in this phase

Experiments – Varied workloads (YCSB)

- Vary parameters every 5 seconds:

  Transactions mix, Percentages of cross-partition transactions, Access skewness (i.e. theta of Zipf)



Experiments – Aggregated Results of Varied workloads (YCSB)

THE UNIVERSITY OF CHICAGO