Adaptive Concurrency Control

Despite the Looking Glass, One Concurrency Control Does Not Fit All

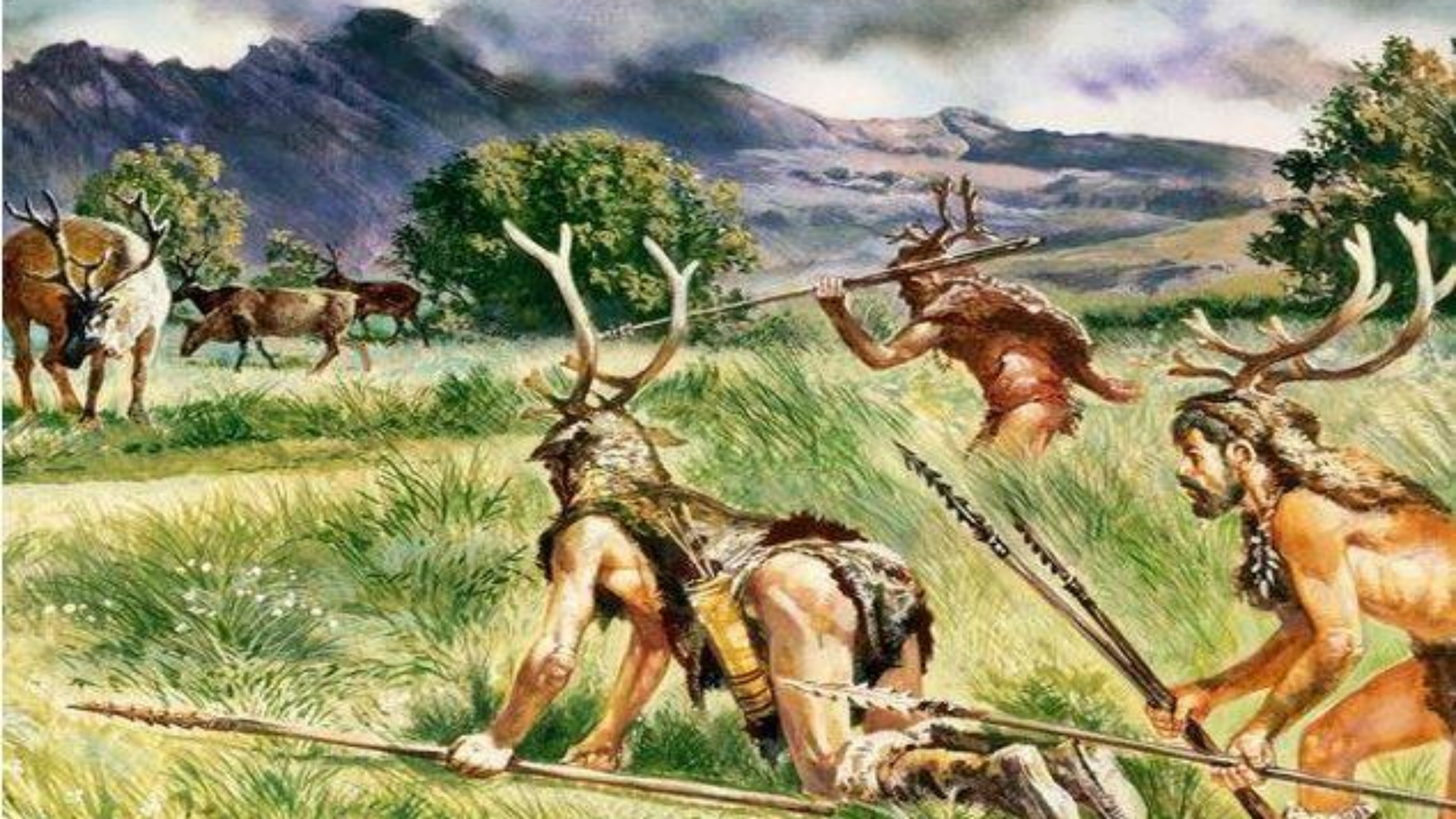
AARON J. ELMORE

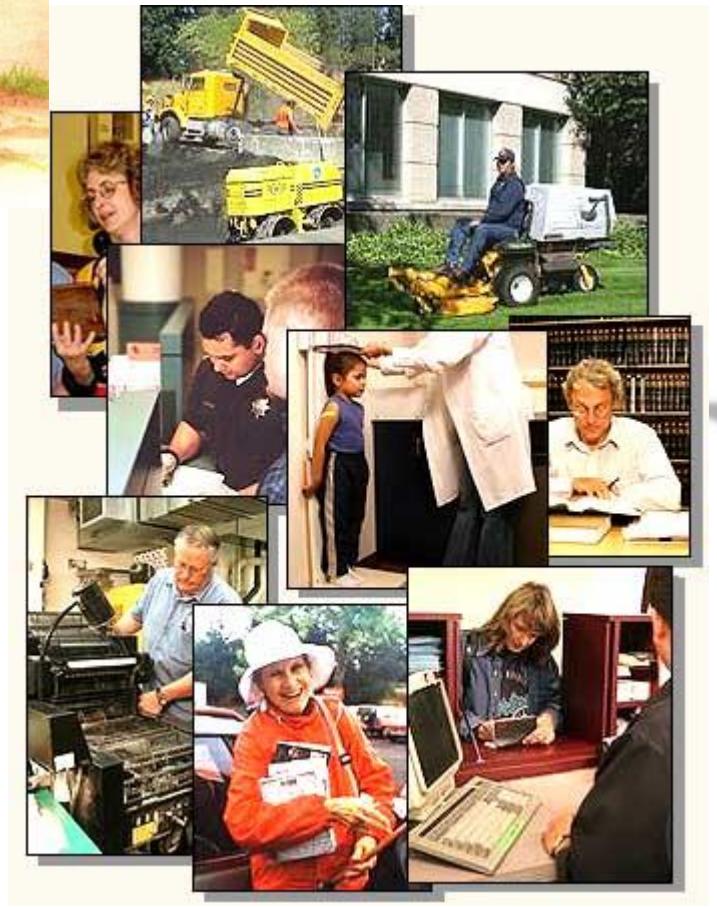
DIXIN "TOTEM" TANG

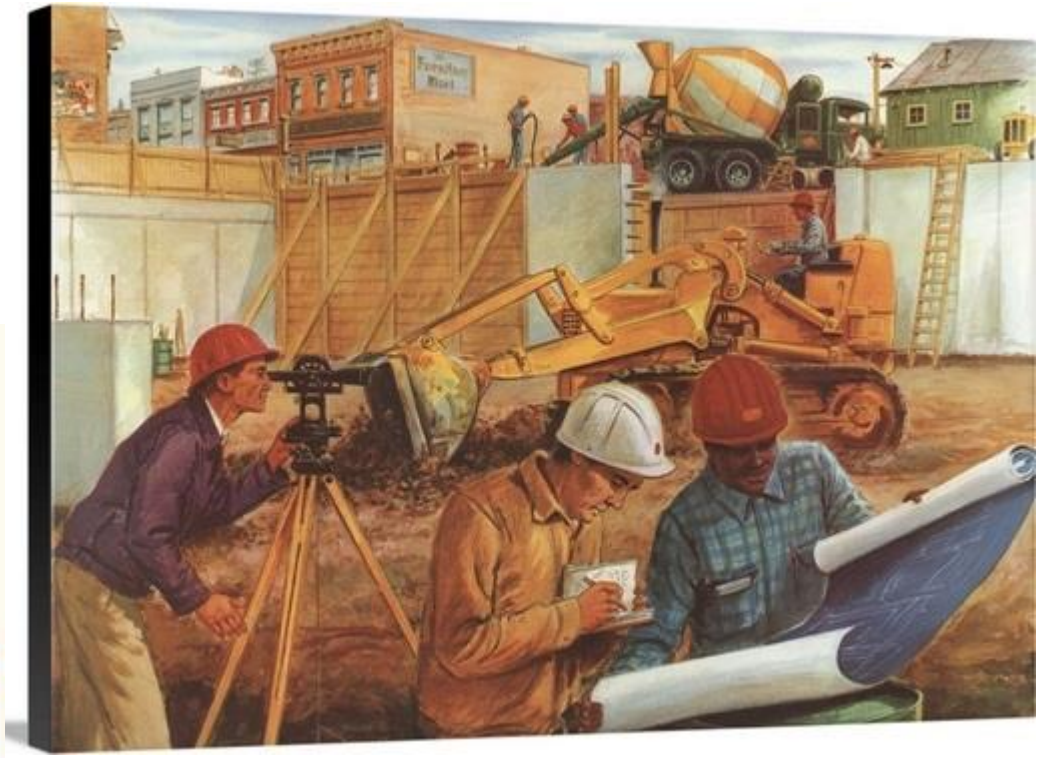
HAO JIANG

THE UNIVERSITY OF CHICAGO


COURT







Choosing the right specialist for a task is important.

WE NEED MORE THAN JUST SPECIALIZATION

Building Many Great “Specialists”

Specialized work in the last decade figuring out how to make OLTP DBs go fast.

H-Store, Hekaton,

Silo, Doppel,

VLL, Orthus, TicToc,

Foedus, MOCC,

[Insert your favorite]


Despite the looking glass,
no one concurrency control protocol can fit all.

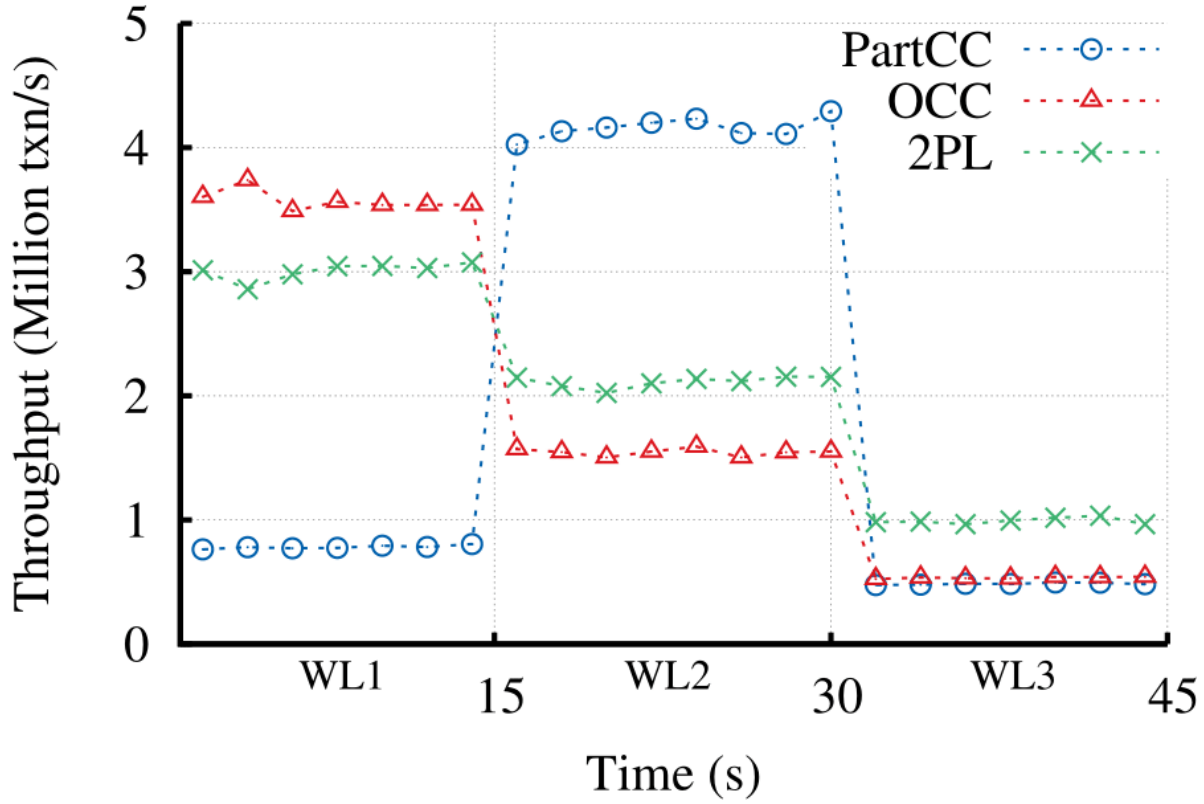
Excel Under the Right Circumstances

H-Store's single threaded (**PartCC**) model suffers with non-partitionable workloads.

Optimistic CC (**OCC**) suffers with high write contention.

Pessimistic CC (**2PL**) is slower than others with low skew.

Sample Workload Variation



YCSB on main-mem DB
OCC ~= Silo
2PL ~= VLL & No-Wait
PartCC ~= Single Threaded

Each CC excels in different scenarios.

WL1: Not partitionable, low skew
WL2: Partitionable
WL3: Not partitionable, high skew

Many have observed this effect to differing degrees, e.g. Carey et al.

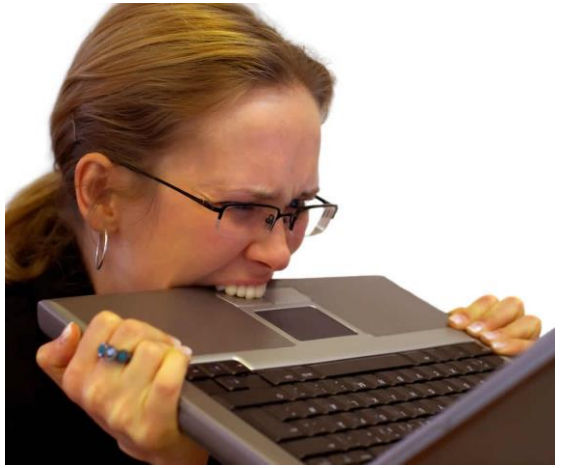
Can we take advantage of specialists (*a CC protocol*) without being brittle to less-than-ideal workload?

Yes we can!
… but which specialists and how many?

Even with Specialists

Specializing is good, but people (DBAs) will struggle to pick the ideal CC.

- Workloads are not know a priori
- Sudden shifts in skew happen
- Gradual changes can occur



Our Vision

Many great static solutions/specialists.

Every ~5 years more self-managed, autonomous, etc. DBMS. We need more than some form of control theory, auto-tuning, or parameter setting.

DBMS systems must be able to adaptively
run with multiple modes/specialists concurrently.

Step towards this Vision: Adaptive Concurrency Control

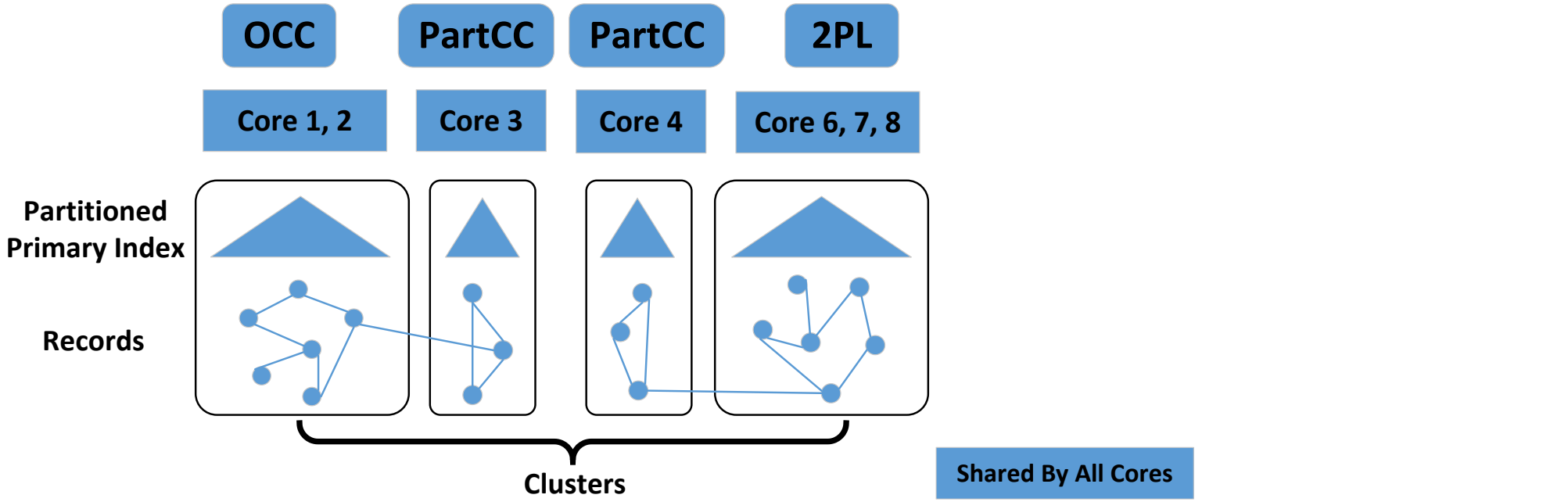
An adaptive and high performance transactional database that:

Clusters data automatically (partitioning with core assignment)

Adaptively selects a concurrency control per cluster

Supports a general framework for mixing CC protocols

ACC Overview

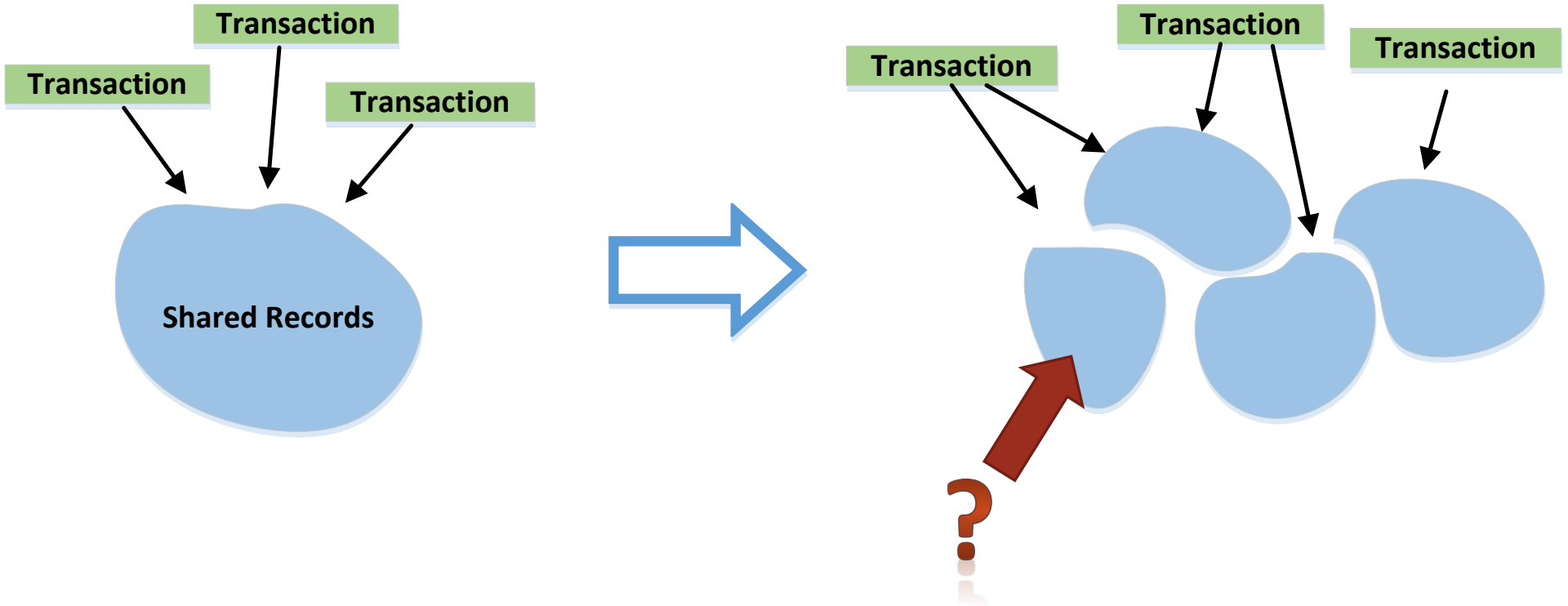


Assumptions:

- Main memory
- Stored procedures
- Deterministic Execution
- Needed clusters are identified by stored procedure params.

Adaptive CC Selection

Choosing a CC Protocol



For now assume we have a good technique to map data into clusters and assign cores / workers.

CC Selection is More than BB ML

Goal: For a given cluster, pick the CC with the highest throughput for a workload

Approach: Learn a model to select right CC (this isn't the 80s)

Challenges:

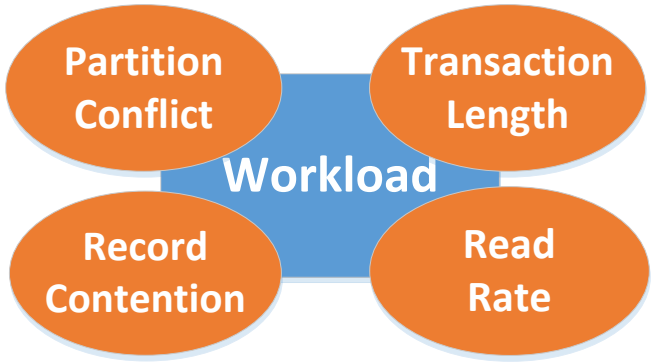
- Feature engineering
- Lightweight feature extraction
- Training models

Workload Modeling

Identify features that have the biggest impact on the comparative performance of various CCs.

Look for signals that impact:

- Partitionability
- Record Skew
- Write Contention

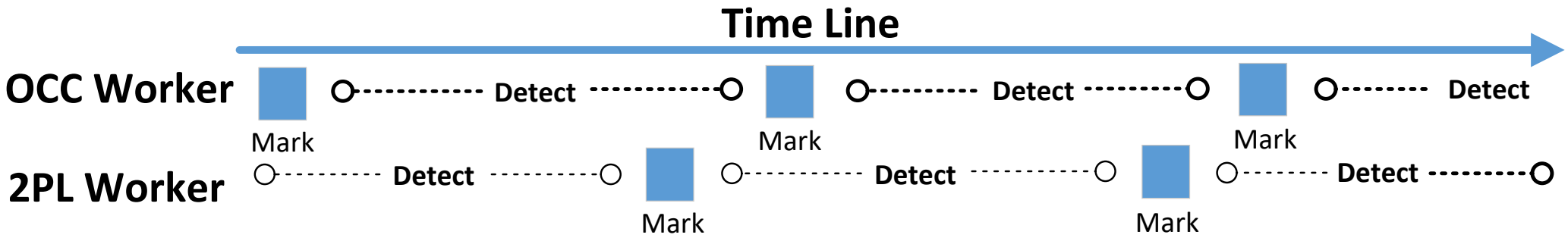


Lightweight Feature Extraction

Feature extraction at scale and across cores is difficult.

- Different CCs exhibit different behaviors
- Centralized solution can be limiting

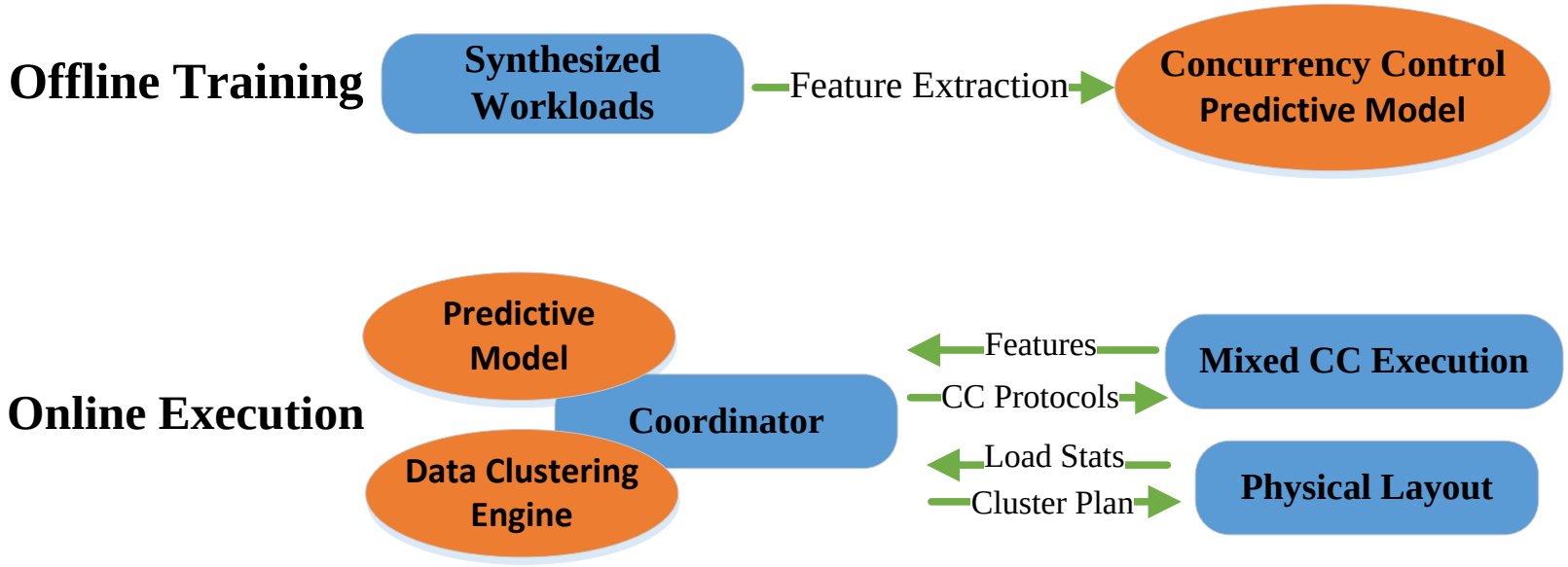
Think about detecting record conflicts / skew across CC



Putting CC Selection All Together

Training the model

Using the model online



Mixing CC Protocols

Mixing CC is More Than Using Locks

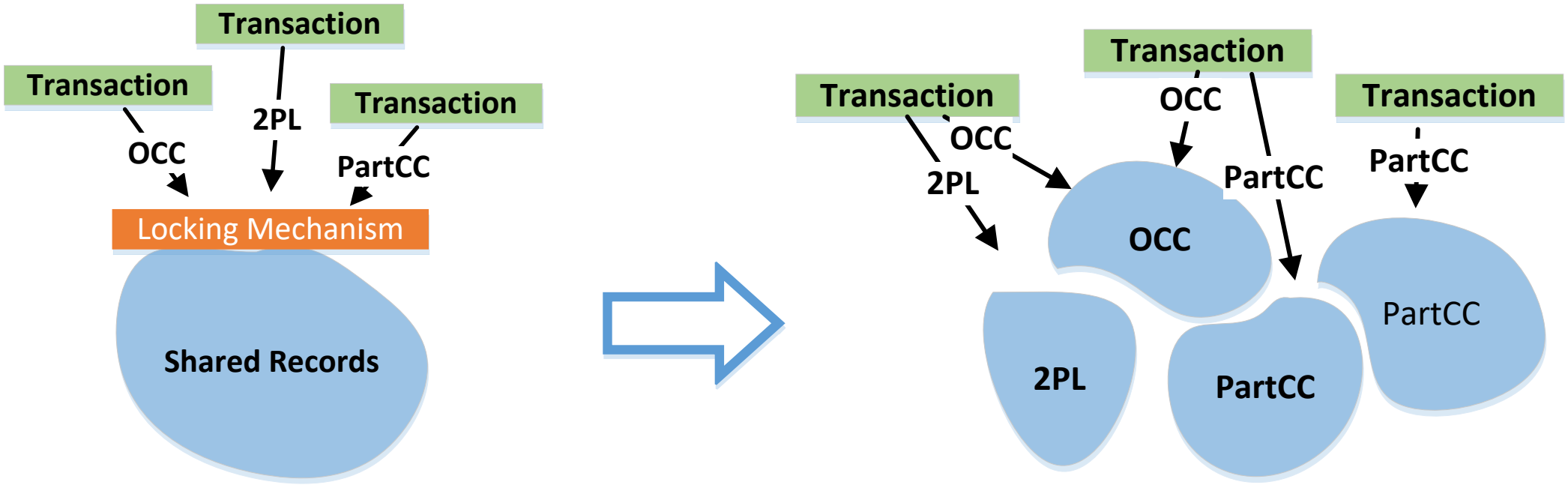
Goal: Support transactions that span multiple clusters with different CC protocols with correctness *and* performance.

Approach: Use the record's "native" CC and span multiple CCs.

Challenges:

- No modification of existing CC protocol
- Lightweight synchronization for conflict detection across protocols
  - (existing approaches rely on centralized or global locking)
- Maintain transaction properties of underlying protocols (e.g. conflict **serializability**, deadlock-free, strictness, etc).

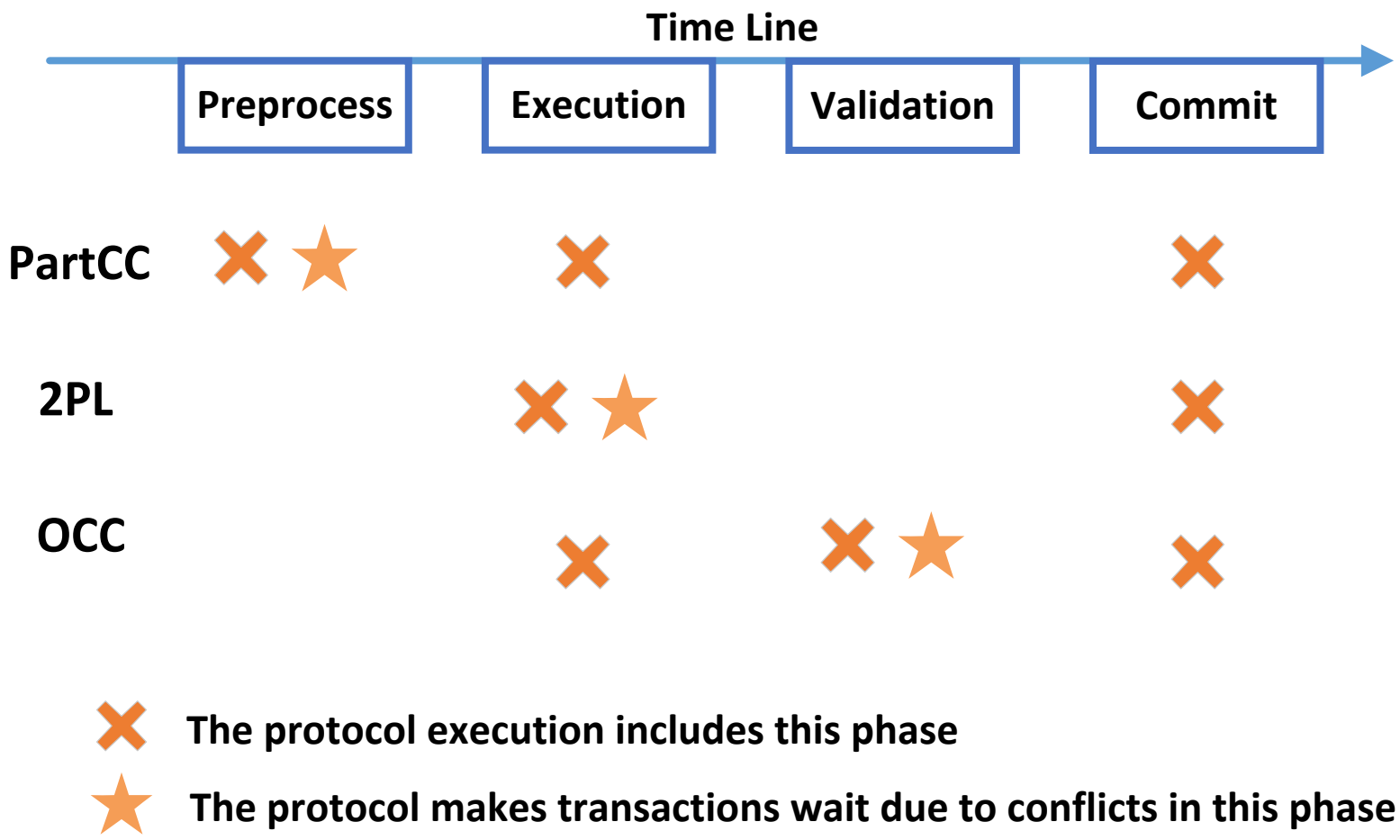
Data-oriented mixed Concurrency Control (DomCC)



Instead of setting global CC

Use the CC “belonging to the tuple”

DomCC for PartCC, 2PL, and OCC



Online CC Reconfiguration

Online CC Reconfiguration

Goal: Allow clusters to change CC without stopping execution.

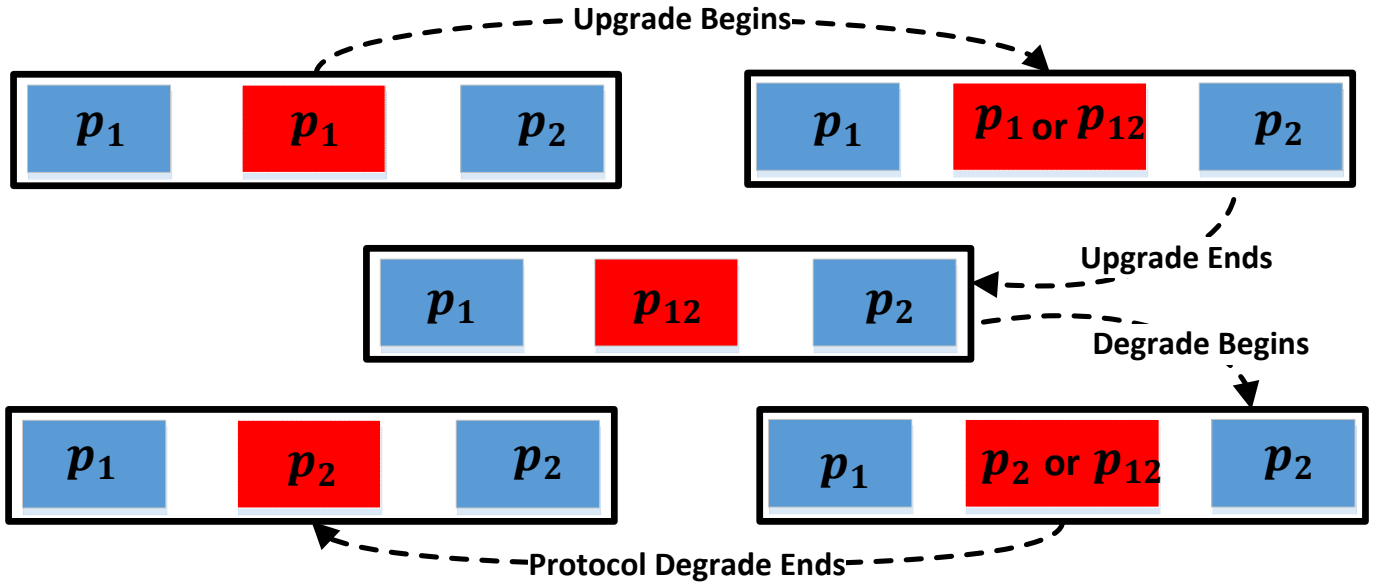
Approach: Allow for hybrid CC during change window.

Challenges:

- Lightweight synchronization for view of CC amongst workers
- General framework for switching
- Partitioned & Non-Partitioned Structures

Online Protocol Switching

Challenge to switch protocols without stopping the entire system or heavyweight synchronization.



Data Clustering

Clustering is More than Partitioning

Goal: Maximize throughput and load balance as secondary

Approach: Holistic approach in progress

Challenges:

- Unknown/flexible number of partitions
- Core allocation (flexible number of workers/capacity)
- Synchronization overhead of CC
- Cost of inter and intra CC distributed TXNs

Lots of Related Work We Skipped Today

Studying CC performance

Hybrid CCs

Optimizing CC

Auto-tuning, self-managed, and adaptive DBs

Hybrid Systems

Rich Set of Future Problems with ACC

Clustering Algorithms

Adaptive Replication

Online Training of Models

Mixed Logging Protocols

Partitioning of Indices (primary and secondary)

Shrinking to Min Capacity

Let’s rethink DBMS architecture to support a mix-n-match approach for components.

…AND FIGURE OUT HOW TO MAKE THEM PLAY NICELY TOGETHER