[OSDI 23`]
ScaleDB: A Scalable, Asynchronous In-Memory Database

Syed Akbar Mehdi, The University of Texas at Austin; Deukyeon Hwang and Simon Peter, University of Washington; Lorenzo Alvisi, Cornell University

Presentation by Minguk Choi

Scalability of Multicore In-Memory Databases

- In-Memory Databases
  - backends for large-scale web applications, public clouds
  - simultaneously write and read intensive
  - both low transaction commit latency and high transactional throughput

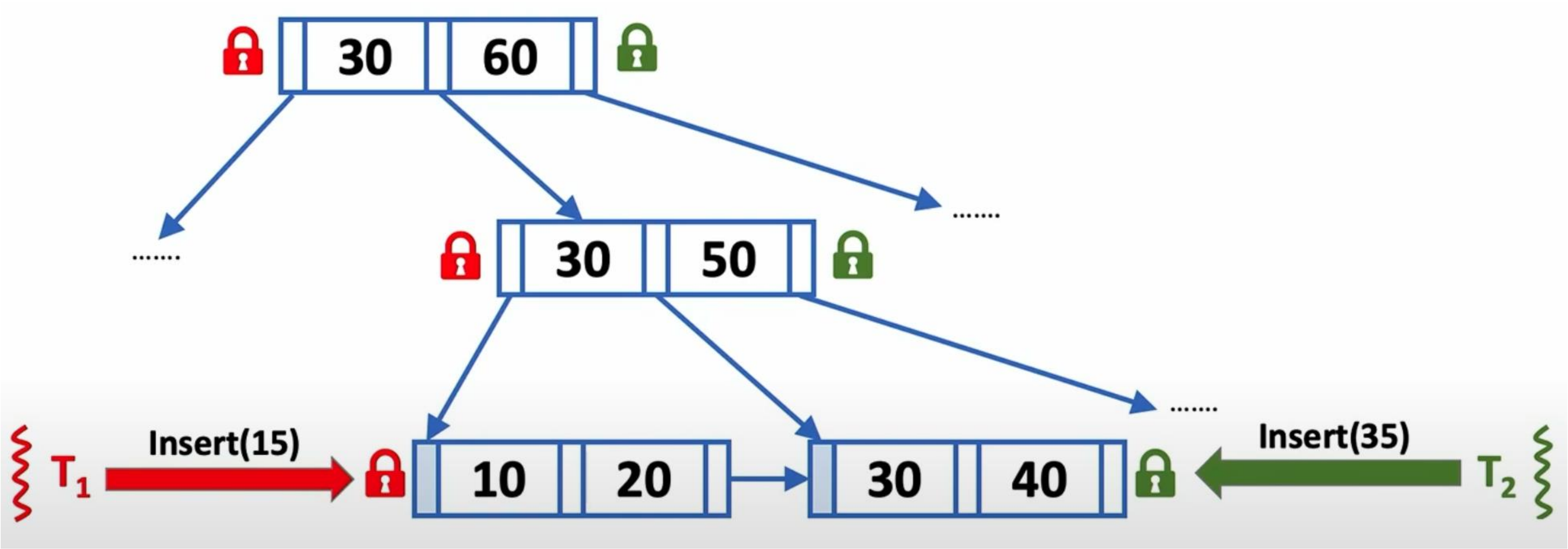
- Yet, database scalability is still limited

Database Scalability Limited by Range Indexes

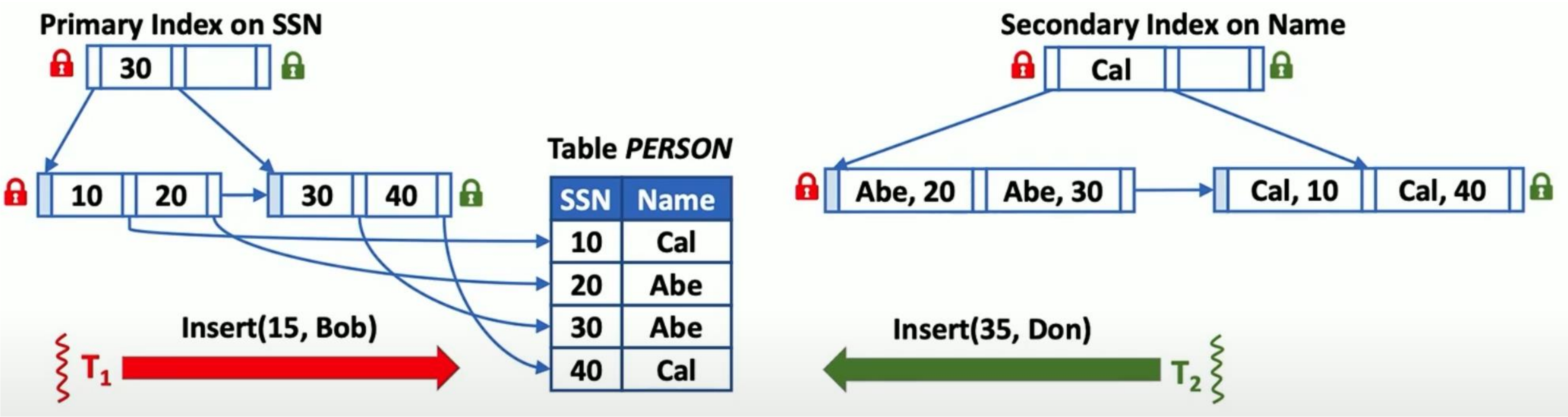
- Range indexes remain difficult to scale
  - B+Tree, ART, MassTree, Skiplist, BwTree, OpenBwTree
  - “under high contention, none of these six data structures perform well”

- Fundamental problem: Hierarchical structure limits scalability
  - But they are required for fast scans in sorted order

Range Index Contention



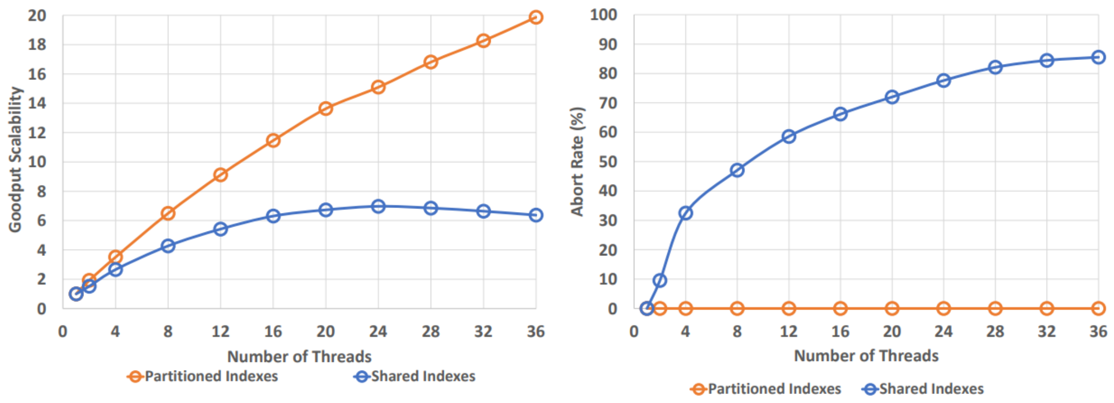
Range Indexes in Context



- Synchronous range index updates → Poor database scalability

- Synchronous range index updates can cause Mechanism Contention
  - Not fundamentally required for serializable isolation guarantee
  - Arising from mechanisms (range indexes) used in database implementation

Range Indexes in Context



(a) Goodput. **(b)** Abort rate.

Figure 2. Cicada scalability on TPC-C ($C_{wh=thd}$) with partitioned and shared indexes.

Benchmark	Read Txns	Range Scans	Database size
TPC-C	8%	7.83%	10 warehouses
SEATS	45%	23%	100K customers
Epinions	50%	100%	200K users

Table 1. Benchmark details.

- Synchronous range index updates → Poor database scalability

- Synchronous range index updates can cause Mechanism Contention
  - Not fundamentally required for serializable isolation guarantee
  - Arising from mechanisms (range indexes) used in database implementation

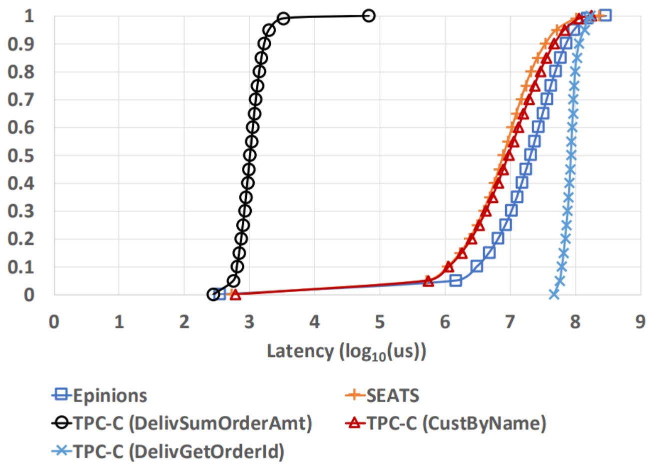
ScaleDB: Beyond Synchronous Range Indexes

- Can we avoid range index mechanism contention to scalably guarantee serializability, with high performance?

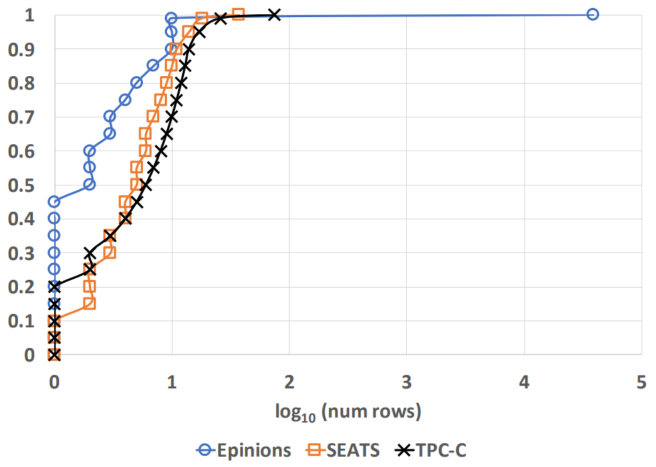
- **Implicit assumption** of prior database architectures
  - Immediately after transaction commit, its writes may be read in a range scan

- But is this assumption exercised in the common case?
  - Experiment to measure **Write-to-Range Scan (W-to-RS)** latency

Long W-to-RS Latencies are Common



(a) Write to range scan (W-to-RS) latency.



(b) Records returned by range scan.

Figure 3. Range scan property distributions of three application benchmarks.

Benchmark	Read Txns	Range Scans	Database size
TPC-C	8%	7.83%	10 warehouses
SEATS	45%	23%	100K customers
Epinions	50%	100%	200K users

Table 1. Benchmark details.

- Reading recently written records → exception for range queries

ScaleDB Design

- Revisit database architecture, not range index scalability
- Design principle: range indexes are asynchronously updated

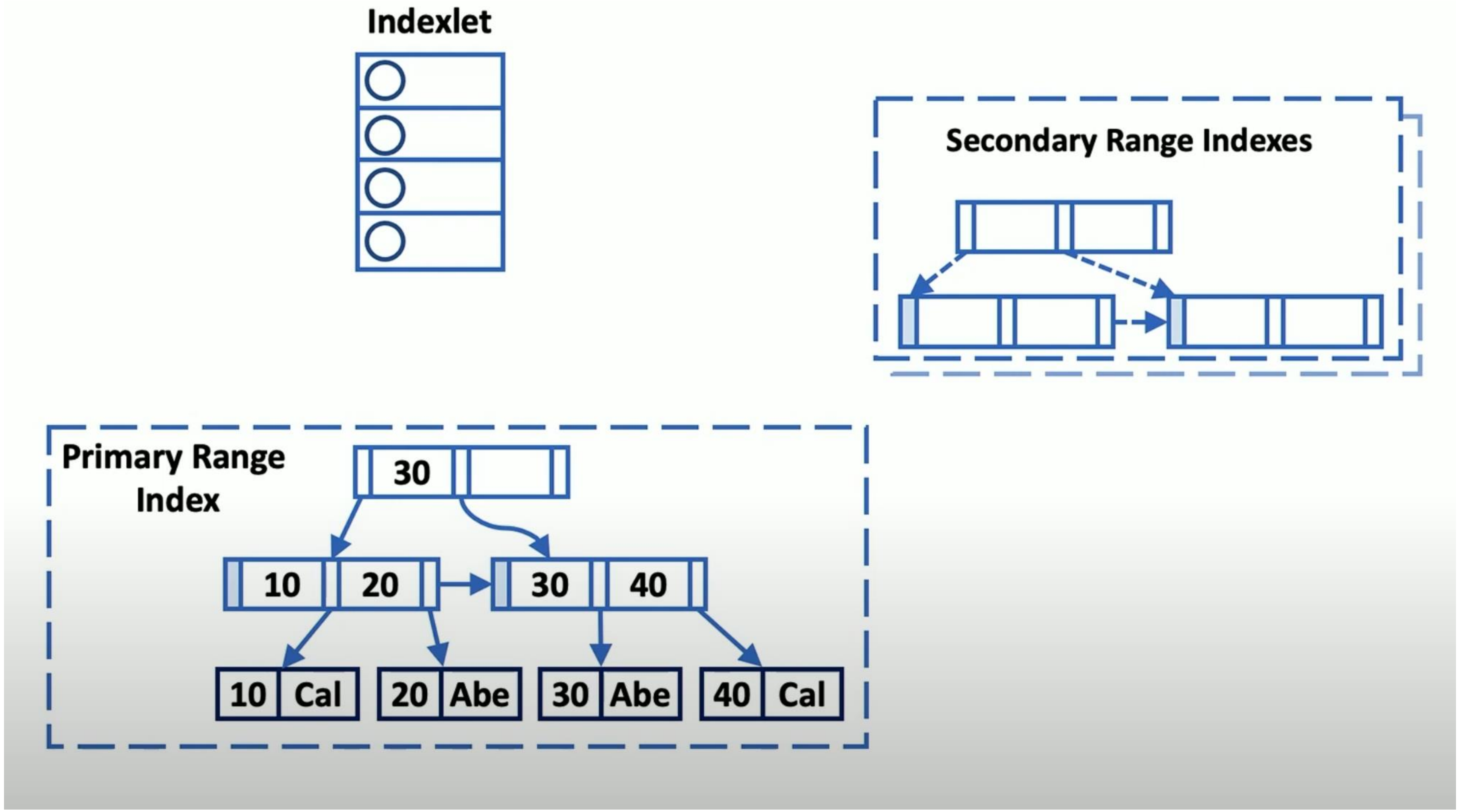
- Two key ideas:
  - Asynchronous range index updates using **Indexlets**
  - **Asynchronous Concurrency Control (ACC)**

Indexlets

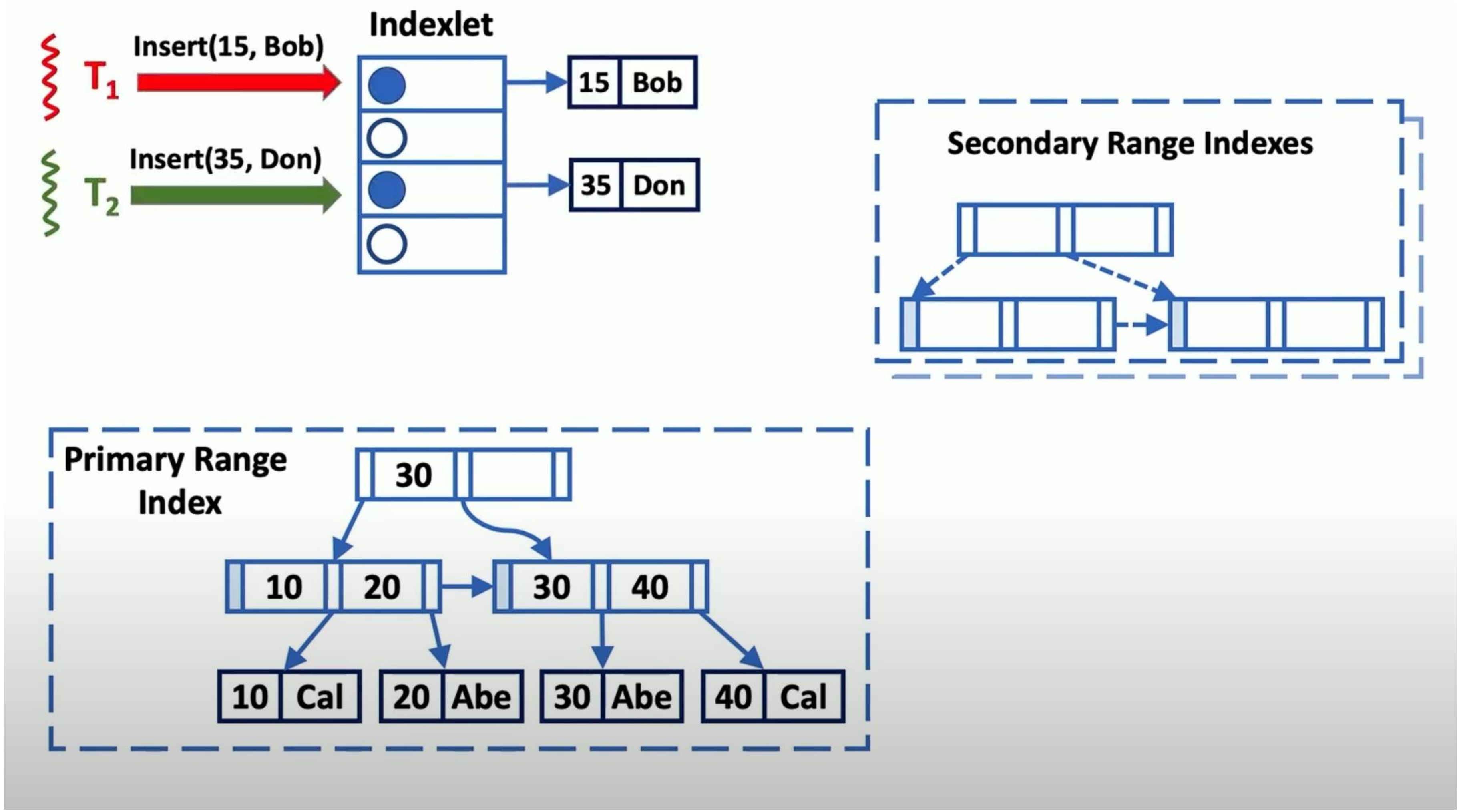
- Hash-based **indexlets** as a temporary store for writes
- No mechanism contention on internal structure
  - Fixed size -> no need to rehash
- One per table (indexlet key = primary range index key)

- Indexlets written synchronously at transaction commit
  - For inserts, updates, deletes
- Periodically flush to range indexes **as a batch** at end of per-thread *merge* epochs

Indexlets

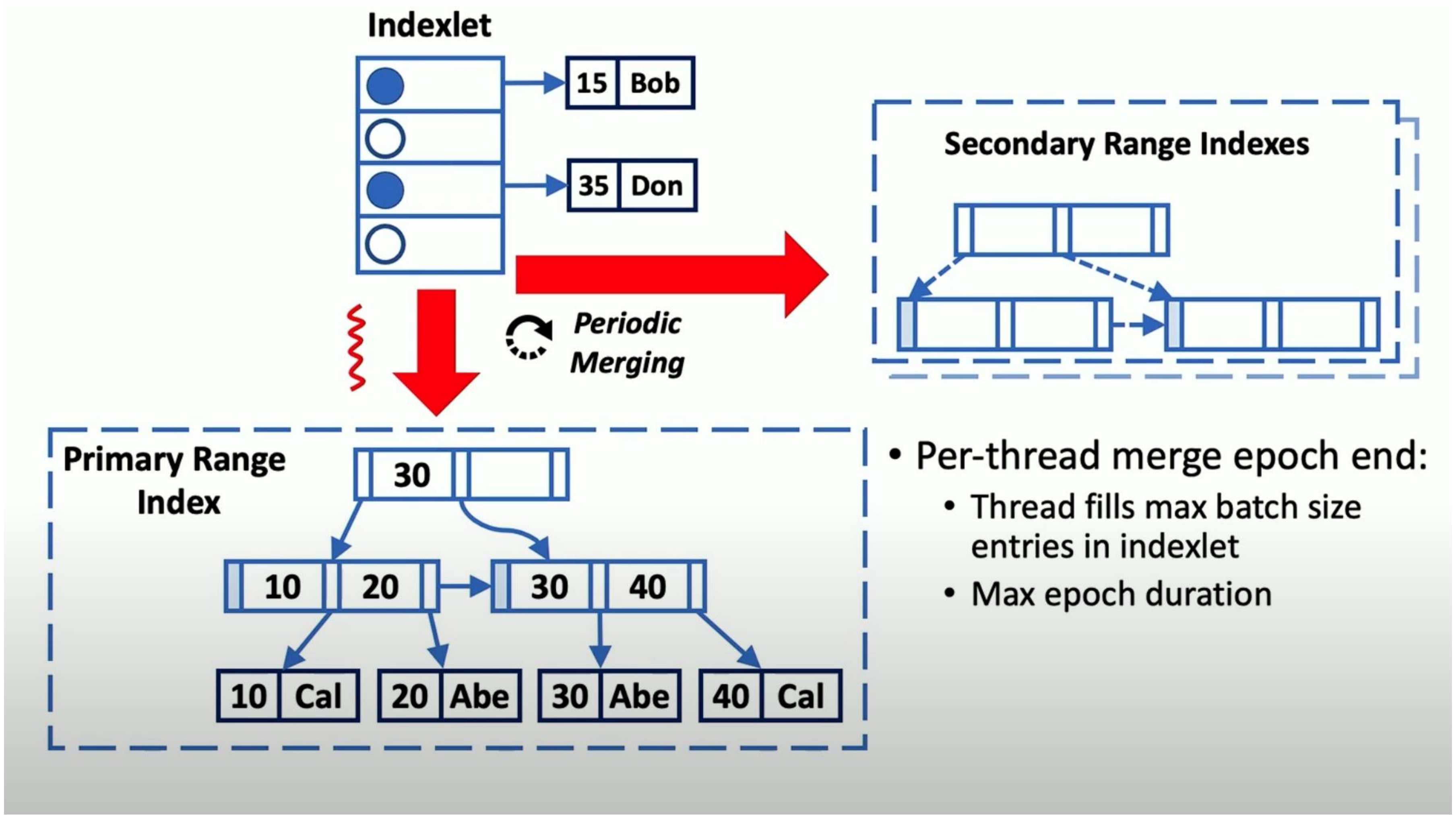

Indexlet
Secondary Range Indexes
Primary Range Index
30
10
20
30
40
10 Cal
20 Abe
30 Abe
40 Cal

Indexlets

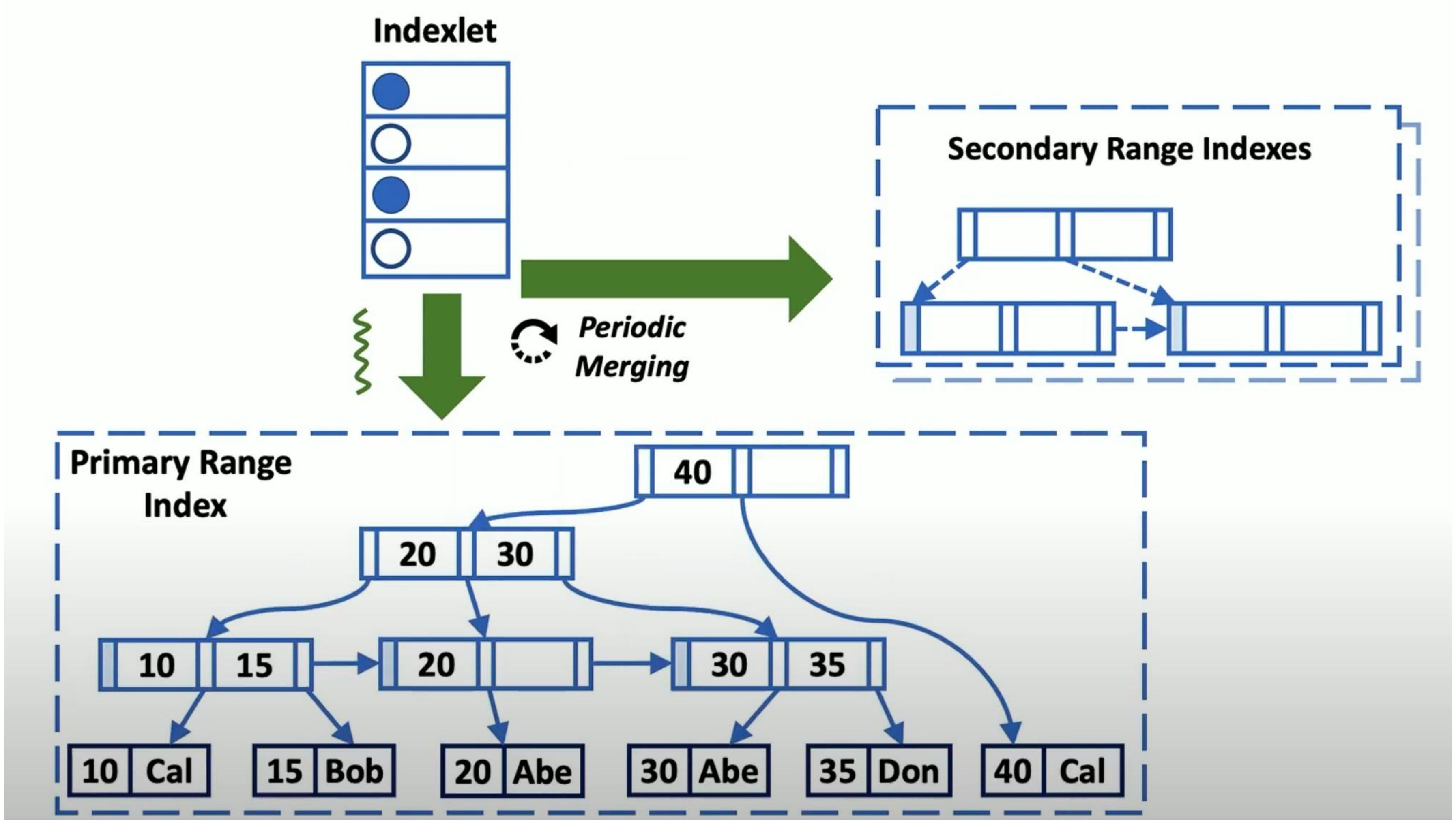


Indexlet
T_1
Insert(15, Bob)
15
Bob
T_2
Insert(35, Don)
35
Don
Secondary Range Indexes
Primary Range Index
30
10
20
30
40
10
Cal
20
Abe
30
Abe
40
Cal

Indexlets


Indexlet
15 Bob
35 Don
Periodic
Merging
Secondary Range Indexes
Primary Range
Index
30
10
20
30
40
10 Cal
20 Abe
30 Abe
40 Cal
• Per-thread merge epoch end:
• Thread fills max batch size entries in indexlet
• Max epoch duration

Indexlets


Indexlet
Periodic Merging
Secondary Range Indexes
Primary Range Index
40
20
30
10
15
20
30
35
10 Cal
15 Bob
20 Abe
30 Abe
35 Don
40 Cal

Asynchronous Concurrency Control (ACC)

- Extends Optimistic Concurrency Control (OCC) with asynchronously updated range indexes

- Point Queries:
  - Search indexlet
  - Not found in indexlet → Search primary range index

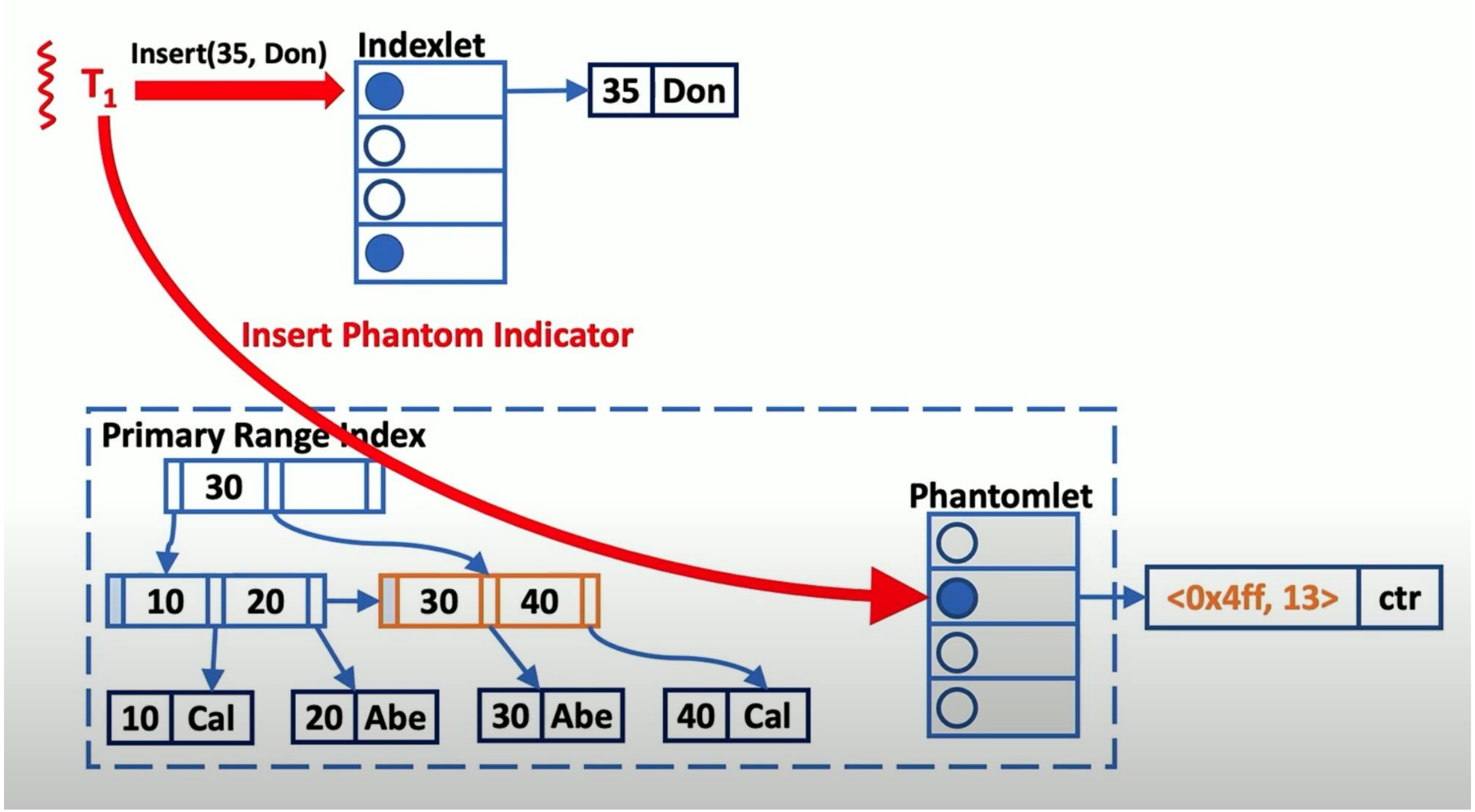
- Range Scans: directly search relevant range index
  - But how to deal with *phantoms*?

Avoiding Phantoms in ACC

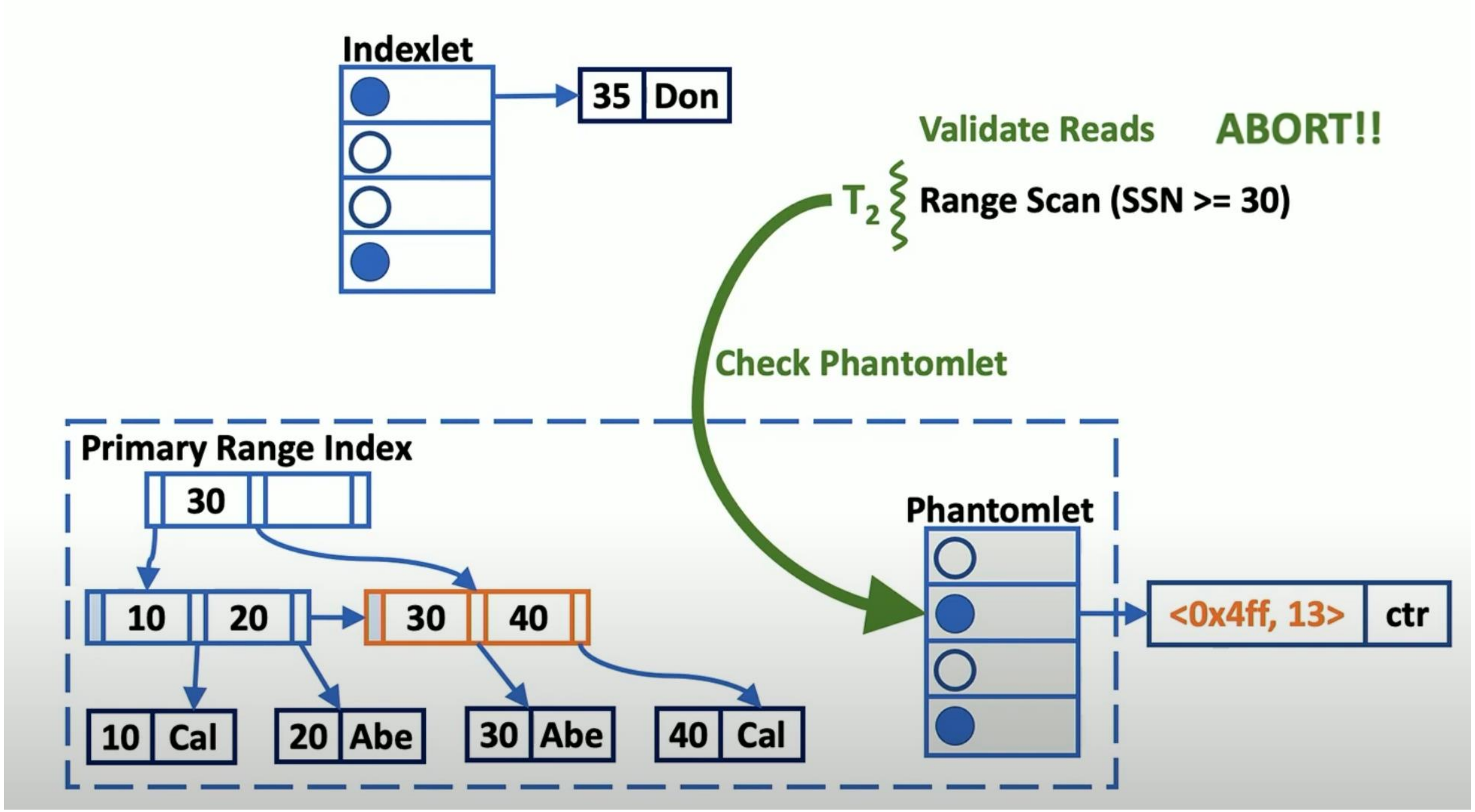
- Phantom: range scan misses a prior committed insert
- Difficult to handle, due to asynchronously updated range indexes

- Solution: **Phantomlets**
  - Phantom detection indexlets
  - Inserting transactions insert **phantom indicators** into phantomlets
  - Range scans validated at commit → Check for phantom indicators

Phantomlets



Phantomlets

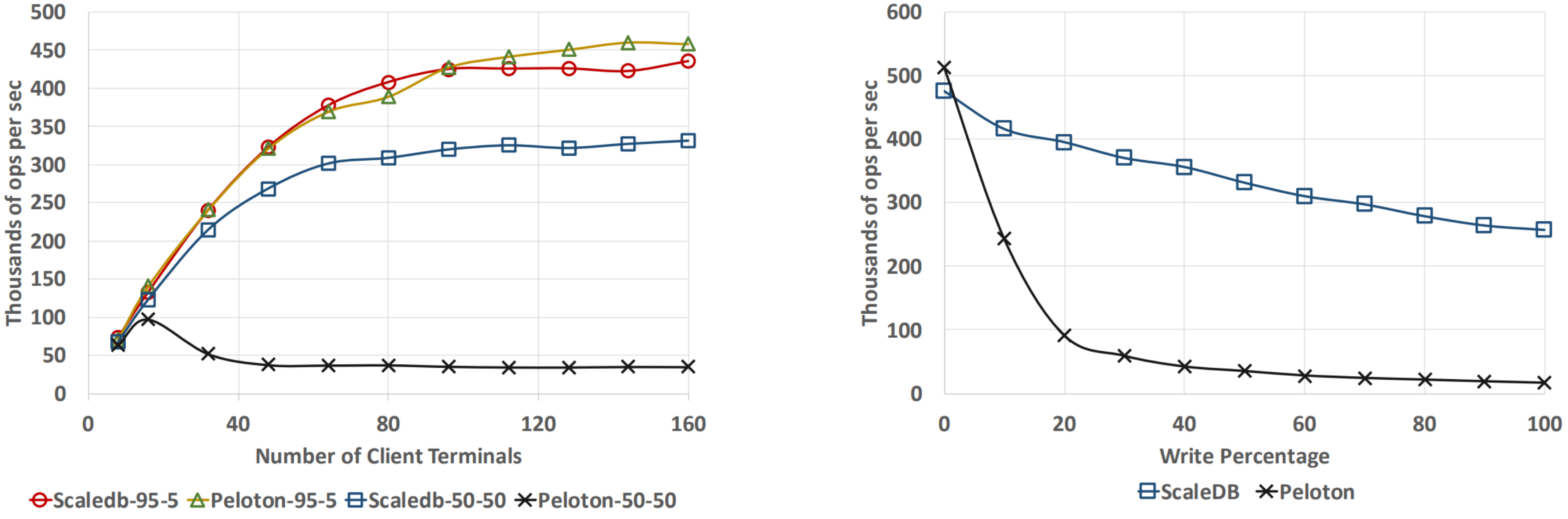


Evaluation

- Compare ScaleDB with Peloton/Cicada
- Goal understand the impact of range index mechanism contention on database scalability
- Evaluate on TPC-C benchmark
  - Configure for low contention (# of warehouses = # of threads)
  - Evaluate using both partitioned and shared range indexes
- Intel Xeon machine with 36 cores (over 2 sockets)

YCSB Scalability

(b) fixing the number
of terminals to 160.

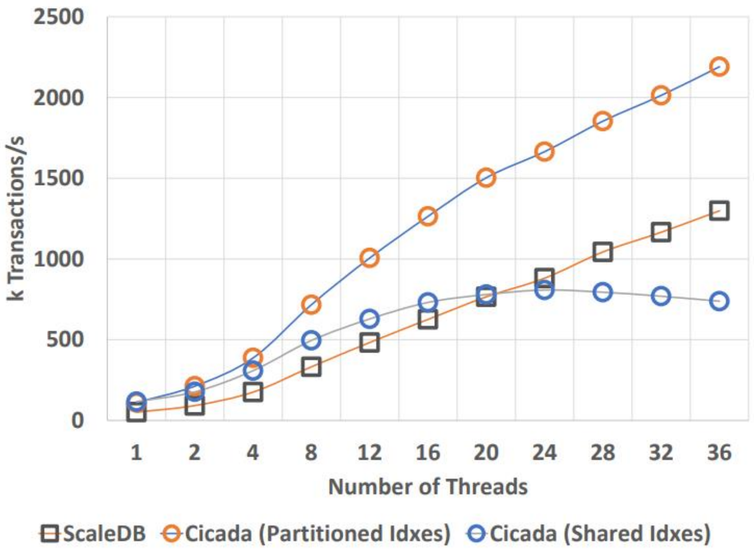


(a) Throughput. **(b)** Write sensitivity.

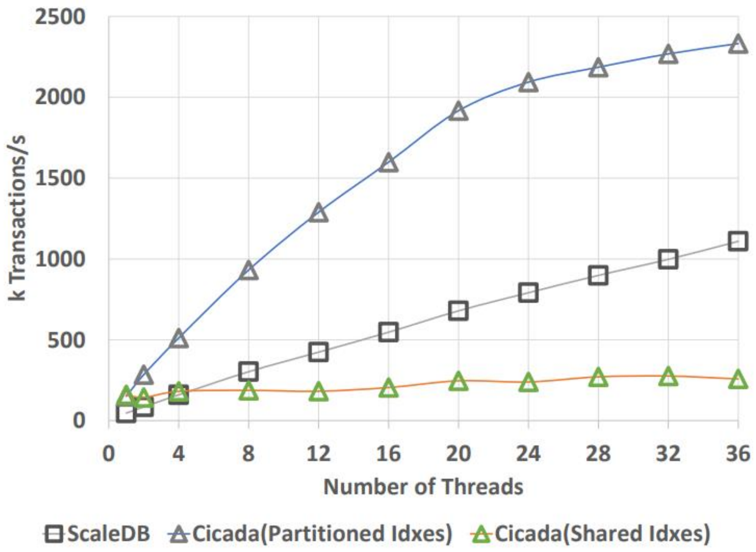
Figure 7. YCSB read-insert workload. 95-5 is 95% reads and 5% inserts. 50-50 is 50% reads and 50% inserts.

- Asynchronous updates to a single range index and compare to Peloton.

TPC-C Scalability



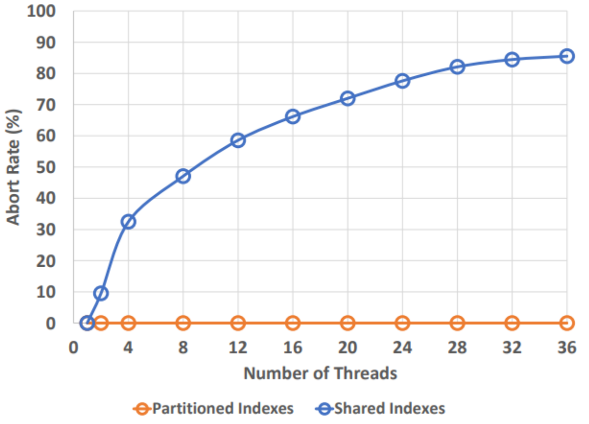
(a) TPC-C.



(b) NewOrd-Deliv.

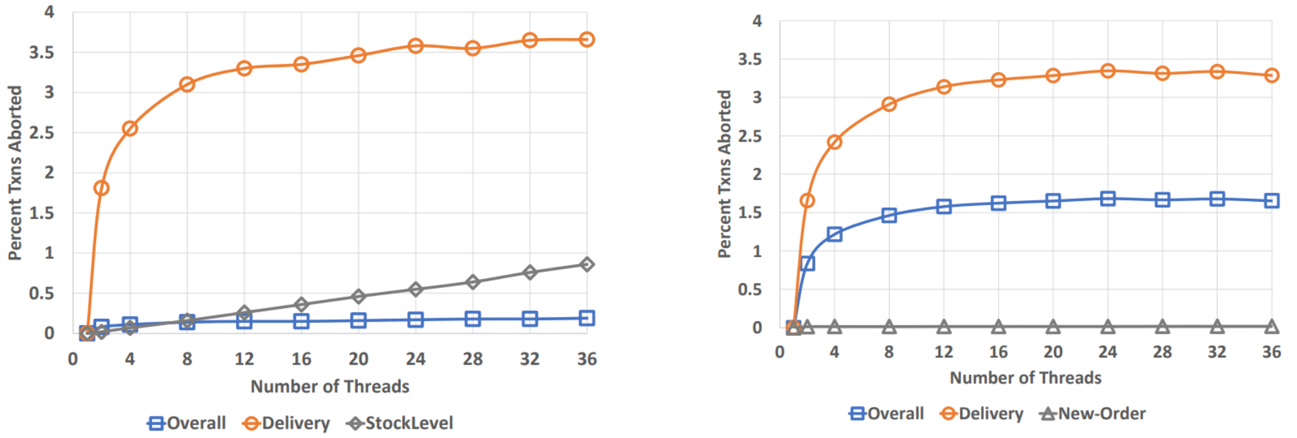
- Asynchronous scalability with serializable transactions on the TPC-C
  - multiple tables and several primary and secondary range indexes.

Abort Rate on TPC-C



(b) Abort rate.

Figure 2. Cicada scalability on TPC-C ($C_{wh=thd}$) with partitioned and shared indexes.



(a) TPC-C. **(b)** NewOrd-Deliv.

Figure 9. ScaleDB abort rate.

Conclusion

- ScaleDB: a scalable, serializable in-memory database
  - Asynchronous architecture to avoid range index mechanism contention

- Two key ideas to build an asynchronous database
  - Asynchronous range index updates using Indexlets
  - Asynchronous Concurrency Control (ACC)

- ScaleDB transcends limitations of a decade of isolated approaches
  - 1.8x better goodput than Cicada on TPC-C