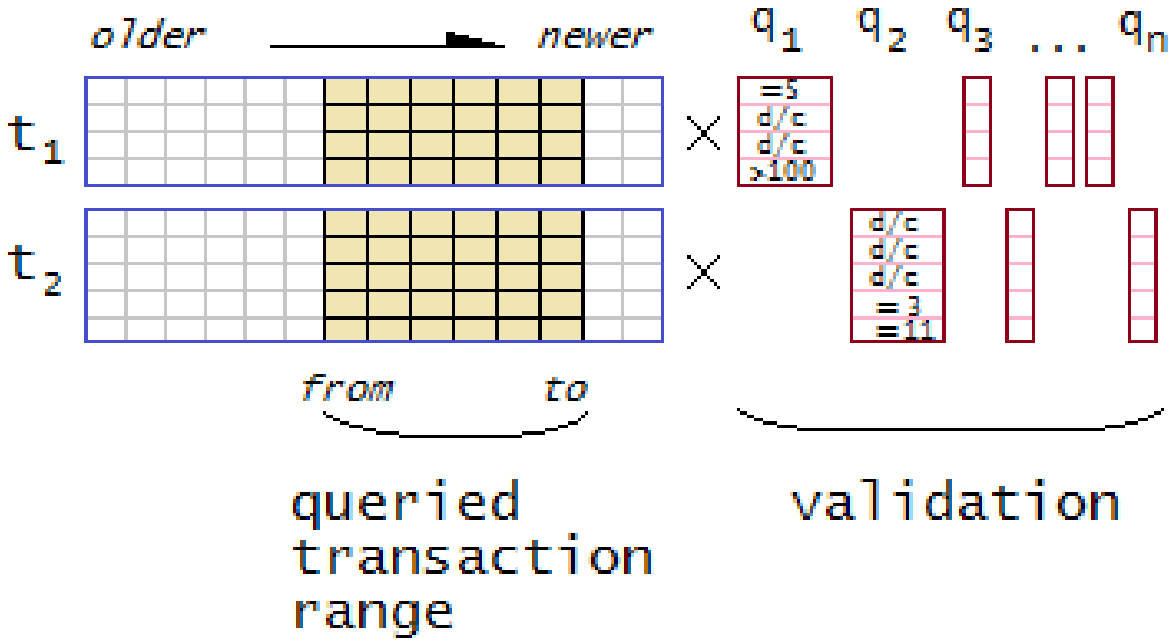
# Validation in Optimistic Concurrency Control

## ACM SIGMOD 2015 Programming Contest

Alexey Karyakin (Crisis)
David R. Cheriton School of Computer Science
University of Waterloo

# The Problem



# Database and Query Statistics I

- Cardinalities
  - » Primary key size, transaction and row insertion/deletion rate
  - » Queries / validation, predicates / query, length of queried txn range, flush batch size
- Probabilities
  - » Satisfied predicates (per type), queries, validations
  - » How often a column is used in queries

# Database and Query Statistics II

- Only recent transactions are validated (~$10^3$ transactions, ~$10^5$ rows)
- $10^3$ validations per batch, 30-50 queries per validation, 10 predicates in each query
- Probability of a satisfied equality predicate is very low: ~$10^{-6}$
- Probability of a satisfied query is also low: $10^{-3}$
  - » Overall, satisfied validation probability: ~5%

# Strategy

- Join ~$10^5$ rows against ~$10^5$ validation queries in each batch
- Build an index on one side and iterate the other side
- Which side (records or validation queries) to index?
  - » Index data and iterate over queries in each batch
- Select the predicate with best selectivity for index lookup
- Some queries cannot use hash index (no equalities)
  - » Resort to table scan

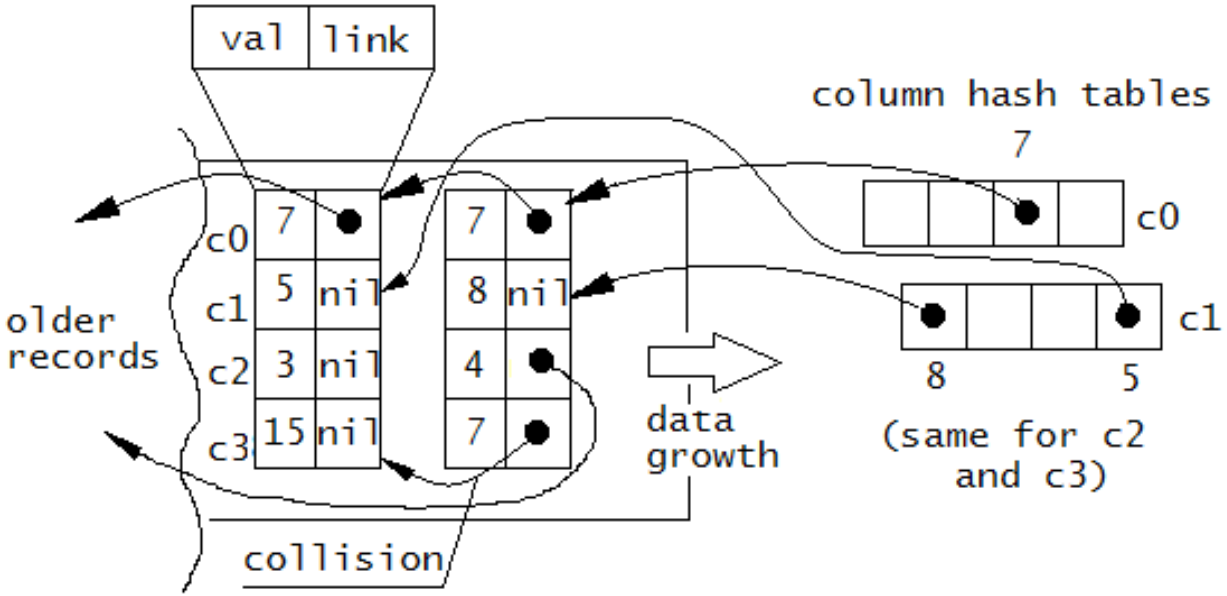
# Data Organization

- Primary key stores existing (not yet deleted) record ids
  - » STX B-tree
- Transaction records are log-structured
  - » New records are appended to the front
  - » Old records are deleted (“forget”) from the back only
- No heap allocation or STL containers

# Indexing

- A hash table is built for each column
  - » Small at start, automatically expanded
- Each record has a link field for each column
  - » Records are chained in historic order
- Looked-up record has to be validated
  - » Non-indexed columns
  - » Hash collisions

# Indexing






UNIVERSITY OF WATERLOO

# Selectivity Estimation

- Only one column may be used for index lookup
  - » Find the field with better selectivity
  - » Some columns have only a few unique values – scan is better
- Column selectivity estimates the number of unique values
- The same hash table is used
- Average allocation distance
  - » The number of records inserted between updating a hash slot

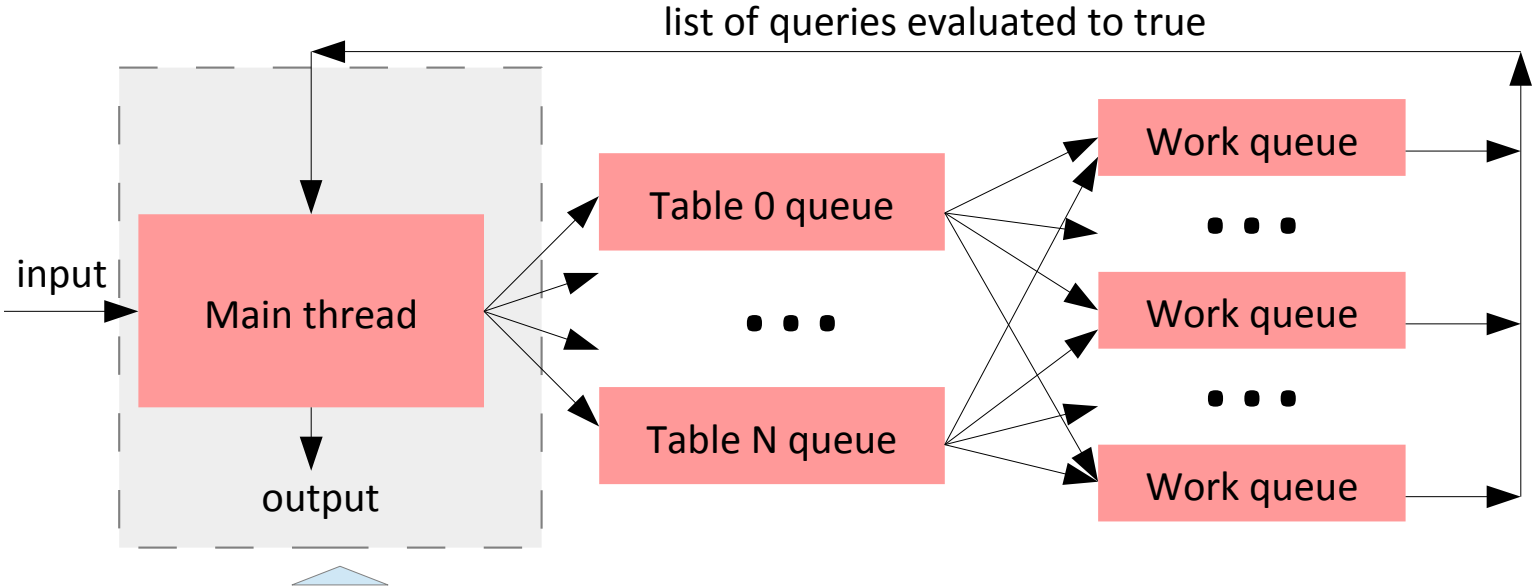
# Table Scan

- Transaction range is scanned backwards, starting with the most-recently inserted record
- Most non-equality predicates (<, <=, >, >=) have high probability of evaluating to true
- However, sometimes the queried value is outside of the range
  - » Min and max values are computed for blocks of records and are used to accelerate the scan

# Parallel Processing

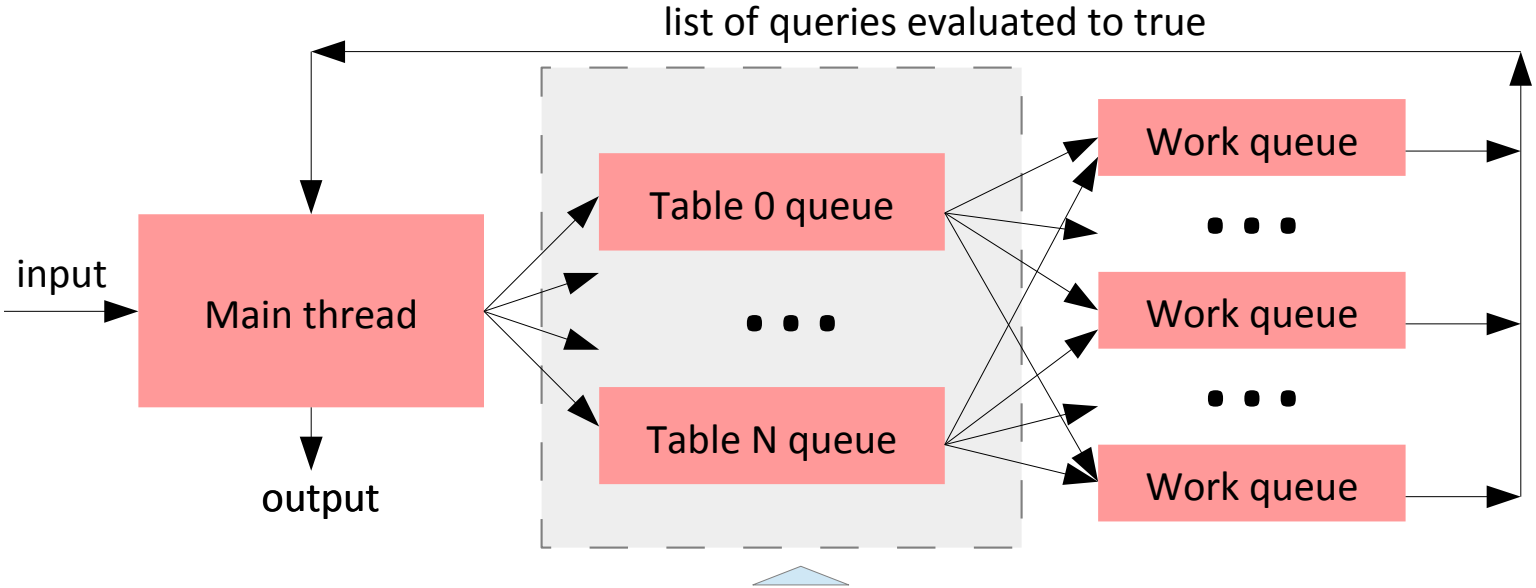
- Data is split into work elements which are transferred between threads via producer-consumer queues
- Three-phase evaluation of each batch
- Small number of threads in the pool
- No locking or synchronization when accessing data

# Data Flow



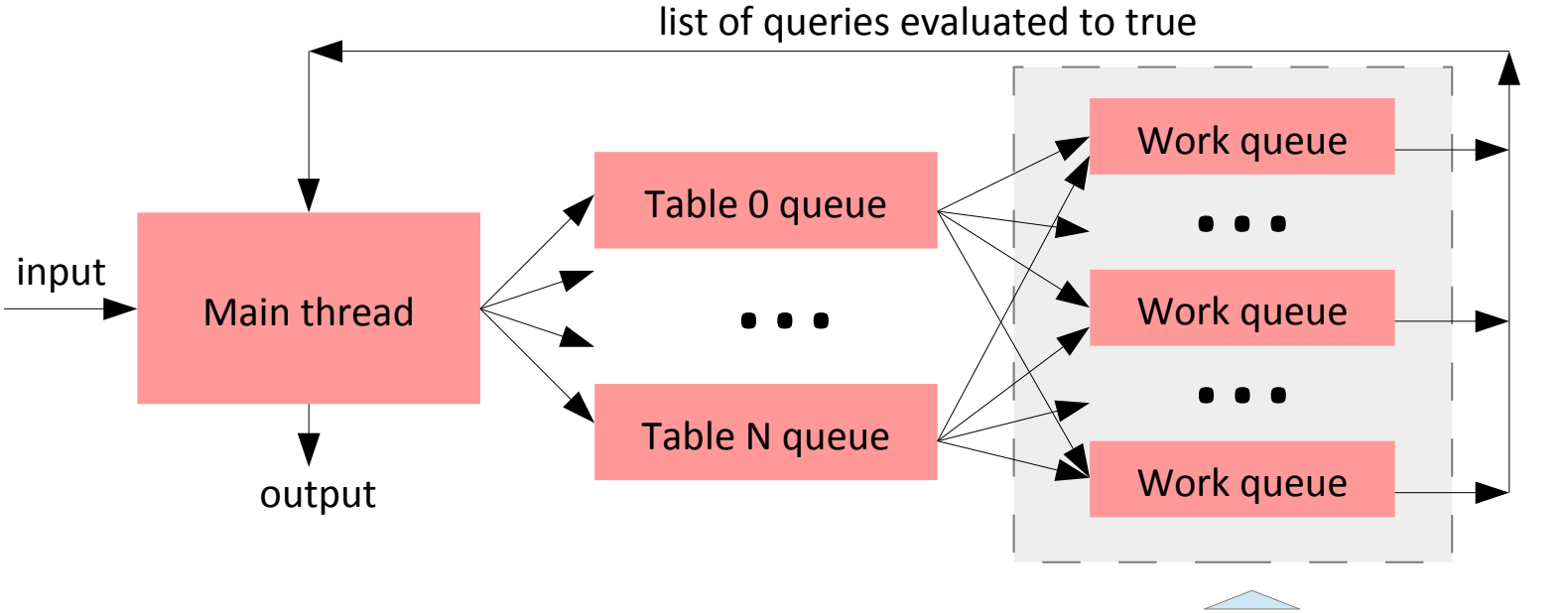

Stage 1: reading and parsing input, distributing data between table queues, batch processing coordination

# Data Flow




Stage 2: updating data and indexes, initial phase of query evaluation

# Data Flow


list of queries evaluated to true
input
Main thread
output
Table 0 queue
Table N queue
Work queue
Work queue
Work queue
Stage 3: complete query evaluation

# Results

- Query evaluation is heavier than data updating
- Most (90%) queries are evaluated using an index
- CPU cache misses is the main performance cost
- The degree of parallelism is low due to high access skew between tables (one table was especially heavy)
- Cost of reading input and parsing is high, compared to actual processing

# Questions

This page intentionally left blank

Thank you.