C-Store: A Column-oriented DBMS

Authors: Mike Stonebraker, Daniel J. Adabi, Adam Batkin,
Xuedong Chen, etc.

Presenter: Ruijia Mao

Why Column Store?

Why Column Store?

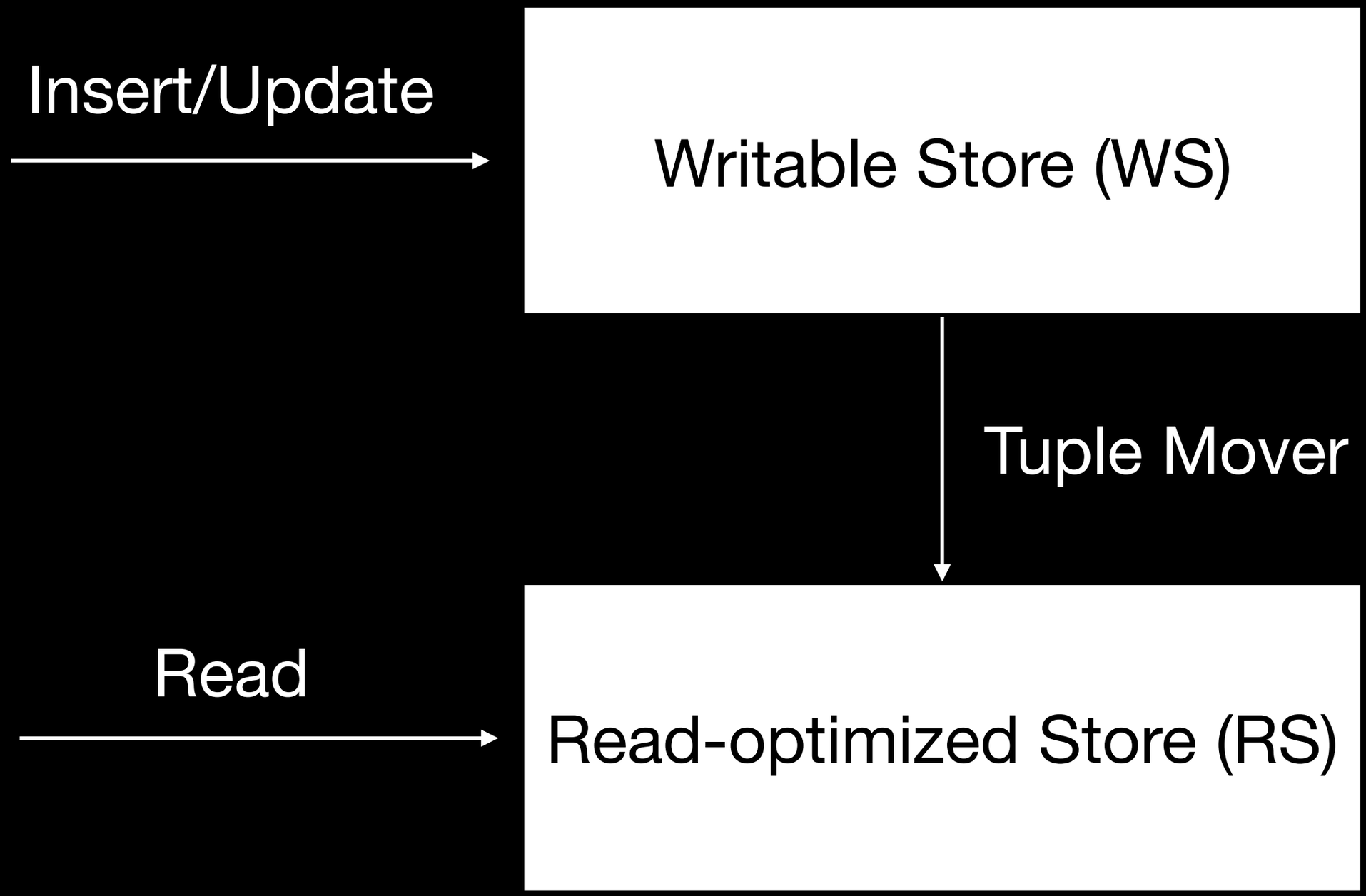
- Avoid bringing irrelevant attributes to the memory
- Use CPU cycles to save disk bandwidth
  - Code data into a more compact form
  - Densepack value in storage
- Other indexes can be applied
  - Bitmap index

Agenda

- C-Store Architecture
- C-Store Data Model
- Read-optimized Store (RS) & Writable Store (WS)
- Updates & Transactions
- Tuple Mover
- Query Optimization
- Conclusion & Comments

C-Store Architecture

C-Store Architecture



C-Store: K-safety

- A system tolerates K failures is called K-safe
- Allocating segments of projections and corresponding join indices into various sites

C-Store Data Model

C-Store Data Model

- C-Store supports relational logical data model
- Physically, C-Store implements projections

C-Store Data Model: Projections

- Projection: a group of columns sorted on the same attributes
- A projection is anchored on a logical table and has the same number of rows as the table

C-Store Data Model: Projection example

Name	Age	Dept.	Salary
Bob	25	Math	10k
Bill	27	EECS	50k
Jill	24	Biology	80k

C-Store Data Model: Projection example

Name	Age	Dept.	Salary
Bob	25	Math	10k
Bill	27	EECS	50k
Jill	24	Biology	80k

EMP1 (name, age)
EMP2 (dept, age, DEPT.floor)
EMP3 (name, salary)
DEPT1 (dname, floor)

C-Store Data Model: Sort Key

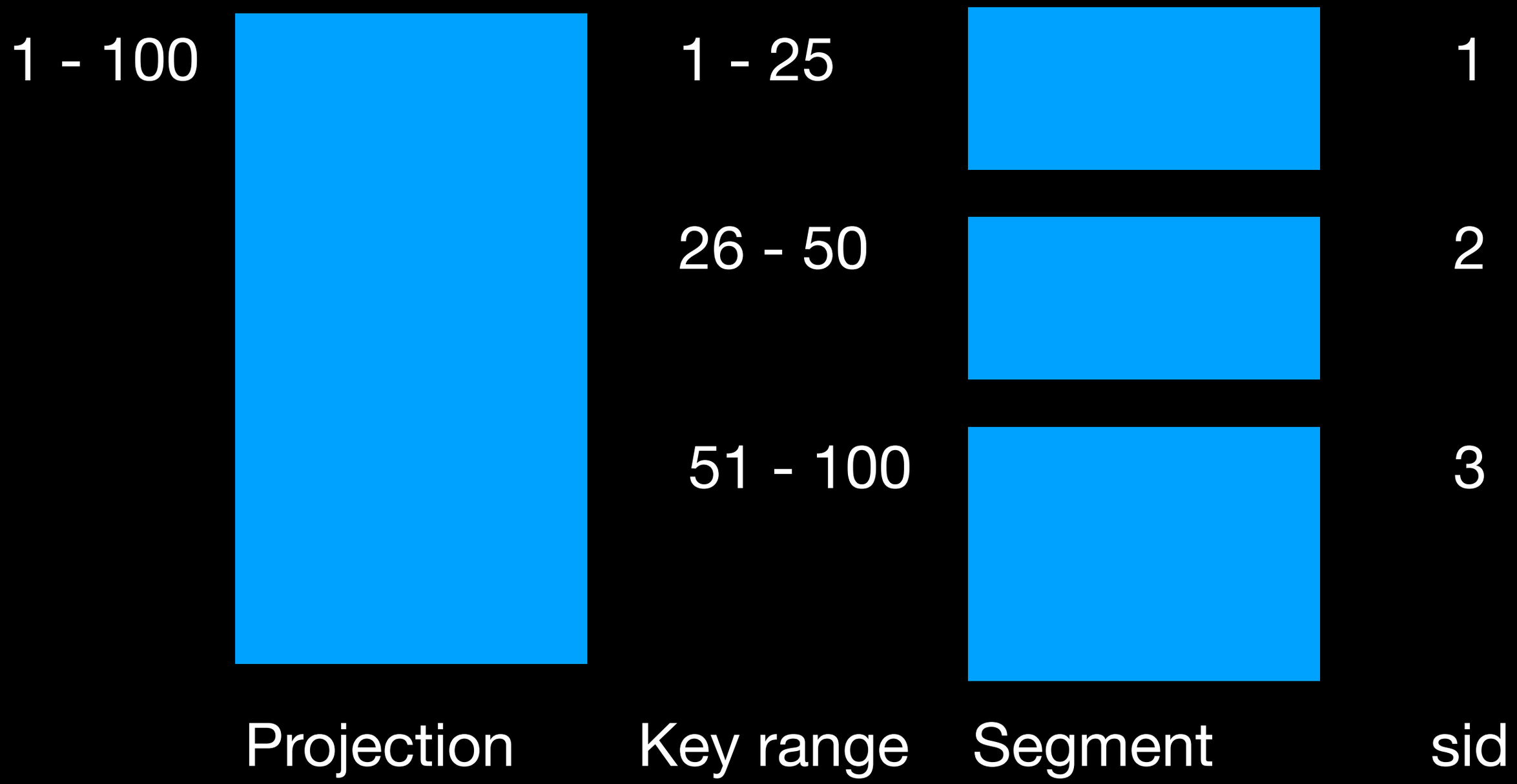
Name	Age	Dept.	Salary
Bob	25	Math	10k
Bill	27	EECS	50k
Jill	24	Biology	80k

EMP1 (name, age | age)
EMP2 (dept, age, DEPT.floor | DEPT.floor)
EMP3 (name, salary | salary)
DEPT1 (dname, floor | floor)

C-Store Data Model: Projection segments

- Every projection is horizontally partitioned into 1 or more segments, which are given a segment identifier
- C-Store only supports value-based partitioning on the sort key of a projection
- Each segment is associated with a ket range

C-Store Data Model: Projection segment example



C-Store Data Model: reconstruct complete row

- Storage key (SK)
- Join indices

C-Store Data Model: reconstruct complete row

- Storage key (SK)
  - Each segment associates every data value of every column with a storage key
  - In RS, storage keys are inferred from a tuple's physical position. In WS, storage keys are physically stored
- Join indices

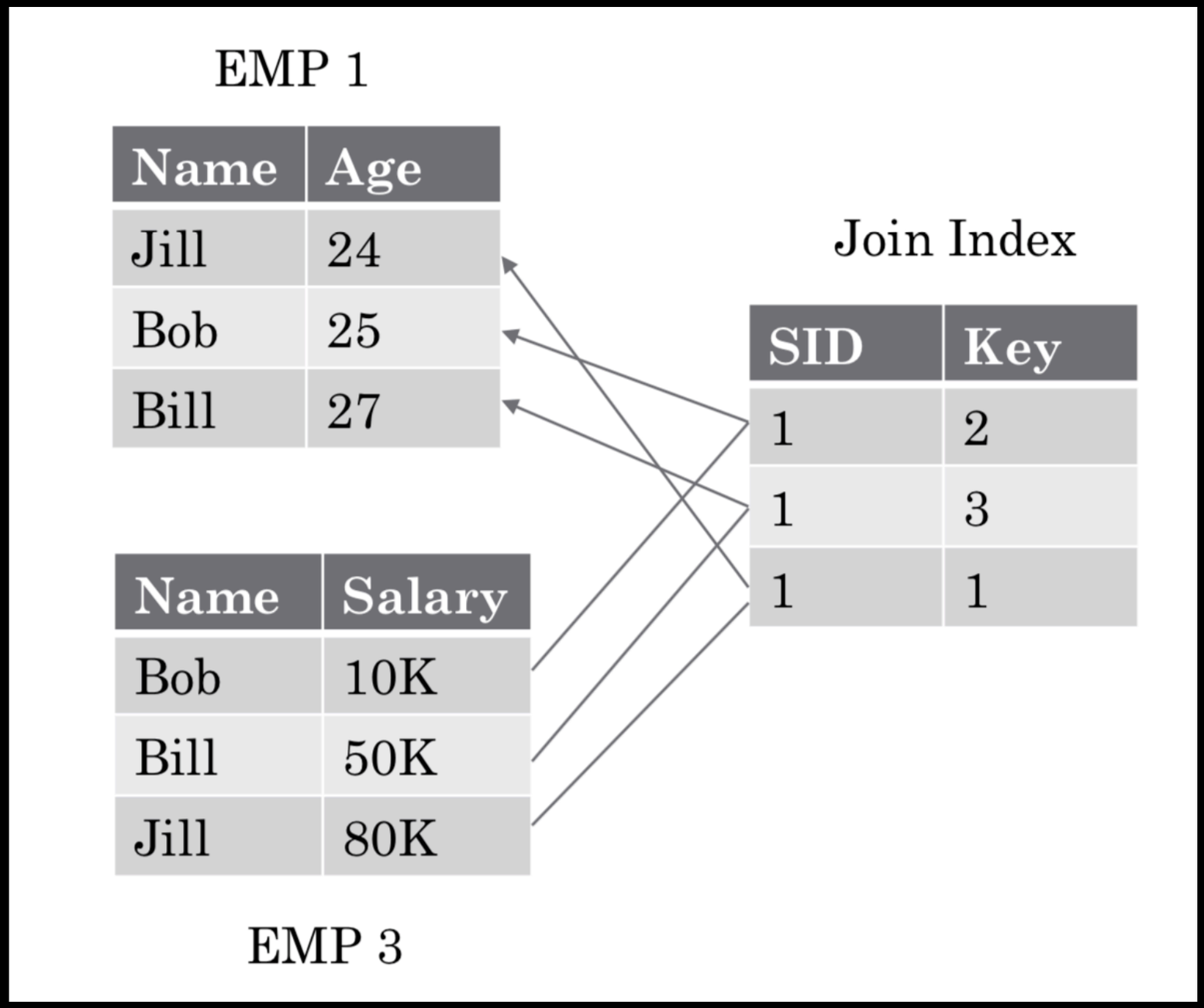
C-Store Data Model: reconstruct complete row

- Storage key (SK)
- Join indices
  - If T1 and T2 are two projections that cover a table T, a join index from M segments in T1 to N segments in T2 is logically a collection of M tables.

C-Store Data Model: reconstruct complete row

- Storage key (SK)
- Join indices
  - For every segment S in T1, the table contains rows of the form:
  - (t: SID in T2, k: Storage Key in Segment t)

C-Store Data Model: join indices example



Read-optimized Store (RS) & Writable Store (WS)

RS

- Columns in RS are compressed using one of four encodings according to their ordering

RS: encoding schemes

- Type 1: Self-order, few distinct values
  - A sequence of tuples (v, f, n)
    - Example: A group of 4’s appears in positions 12-18, represented by (4, 12, 7)
    - B-tree index over v.
    - Can Densepack because no online updates

RS: encoding schemes

- Type 2: foreign-order, few distinct values
  - A sequence of tuples (v, b)
  - Example: A column of integers (0, 0, 1, 1, 2, 1, 0, 2, 1) can be represented as 3 tuples: (0, 110000100), (1, 001101001), (2, 000010010)

RS: encoding schemes

- Type 3: self-order, many distinct values
  - Represent every value in the column as a delta from previous value
  - Example: A column of integers (1, 4, 7, 7, 8, 12) is represented by (1, 3, 3, 0, 1, 4)

RS: encoding schemes

- Type 4: foreign-order, many distance values
  - Currently left unencoded

WS

- WS is also a column store and implements the identical DBMS design as RS
- Storage is drastically different in order to support fast updating

WS: Data Storage

- Storage key, SK, is explicitly stored in each WS segment
- Data don’t get compressed because the size of WS is small
- B-tree index on (v, sk) is used to maintain a logical sort-key order

Updates & Transactions

Updates

- Updates in C-Store are represented as an insert followed by a delete
- An insert is represented as a collection of new objects in WS, one per column per projection, plus the sorted key data structure.

Updates

- Utilize a very large main memory buffer pool to avoid poor update performance.
- Expect “hot” WS data structures to be largely main memory resident.

Update & transaction: Snapshot Isolation

- C-Store isolates read-only transactions using snapshot isolation
- Snapshot isolation works by allowing read-only transactions to access database as of some time in the recent past.
- No lock needed for the past.

Update & transaction: Snapshot Isolation

- High Water Mark (HWM): the most recent time in the past
- Low Water Mark (LWM): the earliest effective time
- Epoch: several seconds, unit of timestamps

Update & transaction: Snapshot Isolation

- No records in RS were inserted after the LWM
- All transactions that start before HWM are complete

Updates & Transactions: Locking-based Concurrency Control

- Read-write transactions use strict two-phase locking
- Write-ahead logging for recovery
- Resolve deadlock via timeouts

Update & transaction: Recovery

- Three types of crashes:
  - Failed site suffers no data loss - execute queued updates
  - Both RS and WS are failed - restore from other sites
  - Only WS fails - most common

Update & transaction: Recovery

- Only WS fails:
  - Find the timestamp for the most recent insertion in RS. If there is no insertion in remote sites, we can simply restore WS with the help of remote sites
  - Otherwise, force the tuple mover to keep log.

Tuple Mover

Tuple Mover

- Move blocks of tuples in a WS segment to the corresponding RS segment, and update join indices

Tuple Mover: MOP

- Tuple mover perform merge-out process on (RS, WS) segment pair
- Locate all records in WS segment with an insertion time <= LWM
  - Those deleted before LWM - Discard them
  - Not deleted or deleted after LWM - Move to RS

Query Optimization

Query Optimization

- Use a Selinger-style optimizer that uses cost-based estimation for plan construction
- Two differences with respect to traditional optimization
  - The need to consider compressed representations of data
  - The decisions about when to mask a projection using a bitstring

Conclusion & Comments

Conclusion & Comments

✓ Column-based storage for efficient read and other smart designs

- WS was not fully implemented at that time

Q&A

Thanks