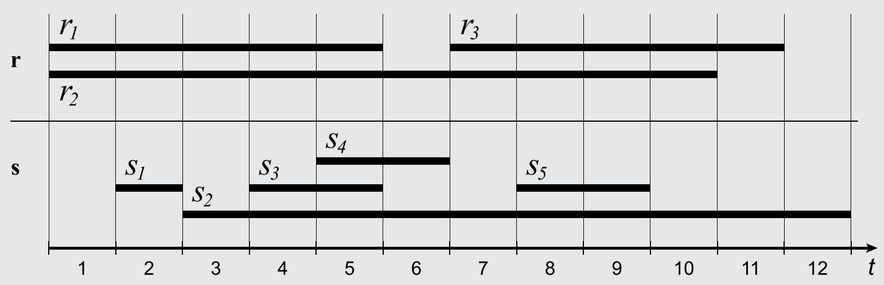
An Interval Join Optimized for Modern Hardware

Danila Piatov Sven Helmer Anton Dignös

Free University of Bozen-Bolzano, Italy

Overlap Interval Join

Problem: Find all pairs of intervals from **r** and **s** that overlap in time.



Answer: $\langle r_1, s_1\rangle$, $\langle r_1, s_2\rangle$, $\langle r_1, s_3\rangle$, $\langle r_1, s_4\rangle$, $\langle r_2, s_1\rangle$, $\langle r_2, s_2\rangle$, $\langle r_2, s_3\rangle$, $\langle r_2, s_4\rangle$, $\langle r_2, s_5\rangle$, $\langle r_3, s_2\rangle$, $\langle r_3, s_5\rangle$.

Very Naïve Solution

Why not use simple SQL?

```
SELECT * FROM r, s WHERE r.Ts <= s.Te AND s.Ts <= r.Te
```

It will work, just very slowly, because it is a join on two independent inequality predicates, and standard RDBMSs are not optimized for that.

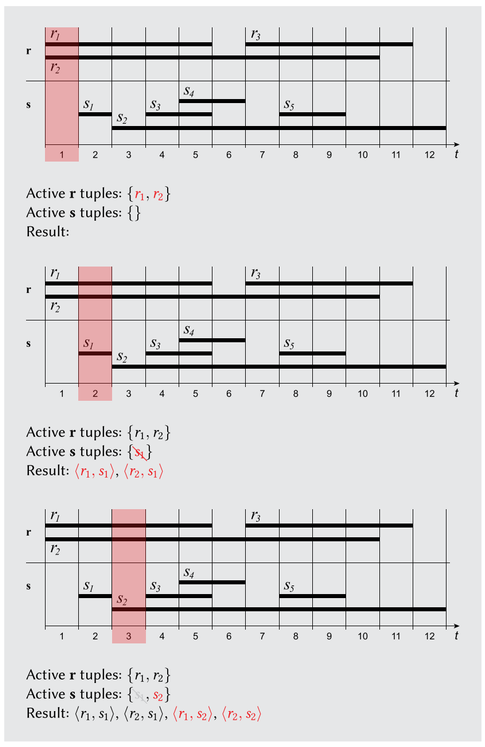
Naïve Solution

Endpoint Index

List tuple events $\langle T_s$, start, *TID*$\rangle$ and $\langle T_e$, end, *TID*$\rangle$ in chronological order. For example, the endpoint index for relation **r** from the above example is: [⟨1, start, 1⟩, ⟨1, start, 2⟩, ⟨5, end, 1⟩, ⟨7, start, 3⟩, ⟨10, end, 2⟩, ⟨11, end, 3⟩]. It can be read as "at time 1 tuple 1 started, at time 1 tuple 2 started, etc."

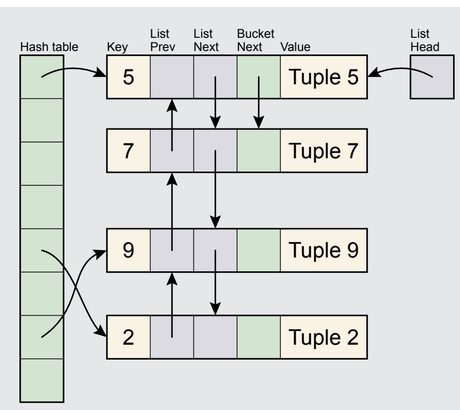
The Algorithm

1) Build endpoint indices for **r** and **s**;
2) Initialize two sets of active tuples (one for each relation);
3) Perform interleaved scan of the indexes (like in sort-merge join);
4) For each encountered endpoint:
   - If it is a left endpoint, load this tuple, add it to the set of active tuples of the corresponding relation and produce cross-product with active tuples of another relation;
   - If it is a right endpoint, remove the tuple from the set of active tuples of the corresponding relation.



Implementation of Active Tuple Sets

Active tuple sets should support tuple insertion, tuple removal by the tuple id and scanning of all tuples. Good candidate is a hash map, but it's not very well suited for scanning. We can connect elements via linked list (like in java.util.LinkedHashMap). This'll give us scanning in linear time with respect to the number of tuples.

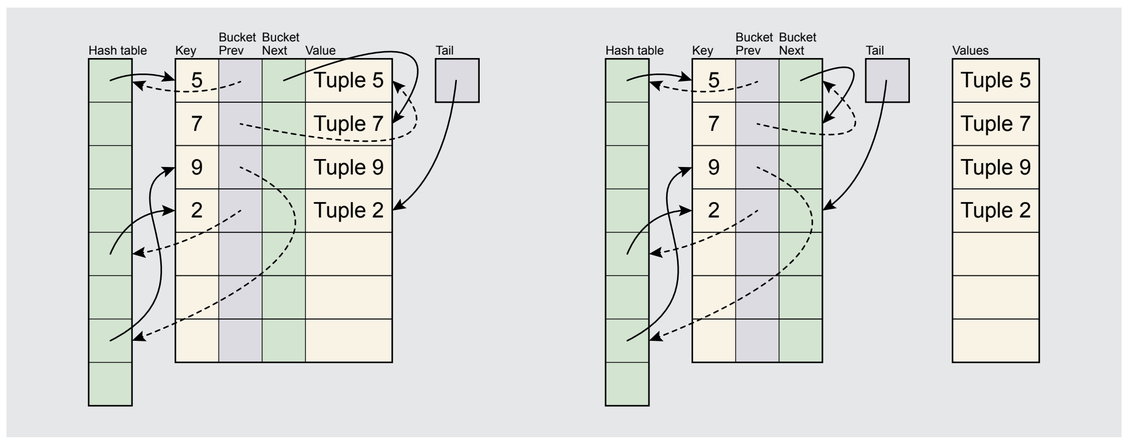


Problem: Random memory access when scanning (up to 200 CPU cycles for one access).

Our Contribution

Gapless Hash Map

Idea: Store items in contiguous memory area. When an element is removed, move the last element to its place. Update all references accordingly (keep back-references for that):



Here normal pointers needed for the hash map are showed as solid arrows, and back-references, needed to update the normal pointers, are showed as dashed arrows. On the right picture values are stored in a separate contiguous memory area under the same indices as the items (i.e. they are logically linked). All item operations are mirrored for values.

Profit: This way we are able to scan hash map values as fast as it is possible—by scanning an array of values.

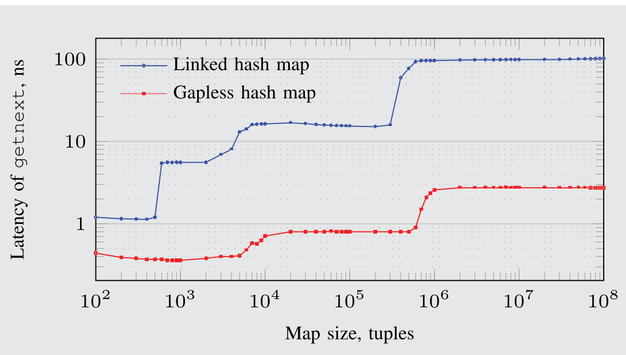
Reducing the Number of Scans

Observation: Often in real-world data multiple tuples start sequentially in one relation while nothing happens in another, causing multiple scans of unmodified active tuple set.

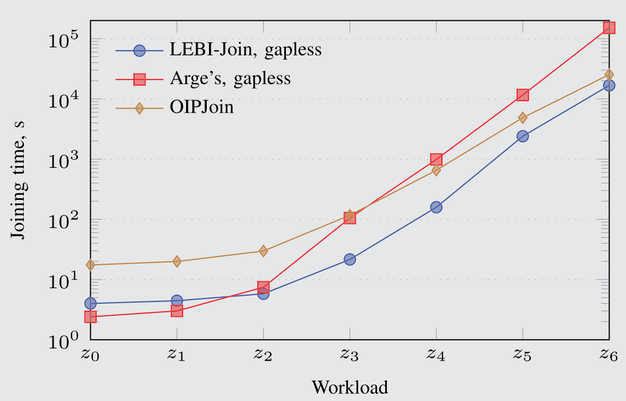
Idea: In such case we can collect all such starting tuples into small array fitting L1 data cache and produce cross-product with the active tuple set by scanning it just once.

Results

Gapless Hash Map vs. Linked Hash Map



Algorithm Comparison Using Synthetic Data



Workload: two relations with 10^6 tules each, varying average tuple length.

D. Piatov, S. Helmer, A. Dignös. An interval join optimized for modern hardware. In *Proc. of ICDE 2016*. Helsinki, Finland.