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 $10^6$ tuples each, varying average tuple length.