transitions. When we have a small number of tuples, all of them fit into the L1 CPU cache and we have a low latency. As the tuple count grows towards 1000 tuples, we start using the L2 cache, which has a greater latency. When we increase the number of tuples further and start reaching 10,000 tuples, the data is mostly held in the L3 cache and, finally, after arriving at a tuple count of around 300,000, the tuples are mostly located in RAM. We make a couple of important observations. First, due to the more compact storage scheme of the gapless hash map, the transitions set in later. Second, the improvement gains of the gapless hash map are considerable and can be measured in orders of magnitude. Third, the latency of a \texttt{getnext} operation for the gapless hash map plateaus at around 2.8 ns, while the latency for the linked hash map keeps increasing, even passing the expected 70 ns. We suspect this is due to TLB cache misses.

We joined the exponentially distributed datasets using different algorithms employing linked and gapless hash maps, the results of which can be seen in Figure 8, showing that gapless hash maps are clearly superior to linked hash maps. The experiments for the dataset “e8” using linked hash maps took too much time and were aborted.

We see that for dataset “e7”, which has active tuple sets containing on average $5 \cdot 10^4$ tuples (fitting into L3 cache), the difference in performance is about an order of magnitude. This corresponds to the performance differences of the \texttt{getnext} operations, confirming our earlier analysis. In the remainder of the evaluation we restrict ourselves to gapless hash maps.

### 7.4.2 LEBI-Join

Next we investigate the performance improvement achieved by using LEBI-Join instead of EBI-Join. Figure 9 shows a comparison for the exponential datasets. The synthetic datasets are unfavorable for LEBI-Join, as they are too uniformly distributed. Nevertheless, from dataset “e4” on, LEBI-Join is able to outperform EBI-Join by a factor of roughly 50% (please note the logarithmic scale).

For real-world datasets the differences can be drastic. Figure 10 depicts the results for the “Incumbent” dataset, showing that LEBI-Join outperforms EBI-Join by a factor of seventeen. Therefore, we drop EBI-Join from the further investigation.

### 7.4.3 Data distribution sensitivity

Another important criterion of operators in database systems is their sensitivity to data distribution, as this makes it harder for query optimizers to obtain reliable statistics for building stable execution plans. Figure 11 shows the behavior of the different algorithms when facing uniformly and exponentially distributed datasets. We can clearly observe that the sort-merge algorithm reacts very sensitively to changes in data distribution and OIPJoin is affected to a minor extent, while the other algorithms are unaffected.

### 7.4.4 Comparison to the State-of-the-art

Having identified the strongest version of our algorithm, we now compare it to the state-of-the-art, namely Arge’s interval join, OIPJoin, and sort-merge join.