HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

FUSED TABLE SCANS: COMBINING AVX-512 AND JIT

Markus Dreseler, Jan Kossmann, Johannes Frohnhofen, Matthias Uflacker, Hasso Plattner

Joint Workshop of HardBD & Active @ ICDE
Paris, 16th of April 2018

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

FUSED TABLE SCANS – COMBINING AVX-512 AND JIT

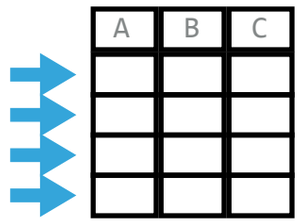
- AVX-512: Intel's newest instruction set for SIMD operations
- **J**ust-**I**n-**T**ime compilation: Creating binary code at program runtime

- Efficient (multi-predicate) sequential scans are a necessity for relational database systems
- Secondary indexes can speed up such operations
  - Drawbacks: memory consumption and maintenance cost
- **Contribution**: Combine the above techniques to accelerate table scans

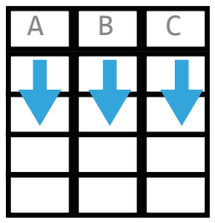
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

FUSED TABLE SCANS – COMBINING AVX-512 AND JIT

- Optimizations of sequential scans can be grouped into two categories
  - Block-at-a-time: Evaluate multiple values (SIMD) of a column at a time
    - Store results in position list
    - Materialization between operators



- Data-centric compilation: Generate (JIT) a tight, optimized loop to process one tuple at a time
  - No utilization of SIMD until now
  - Suboptimal interplay with some hardware optimizations



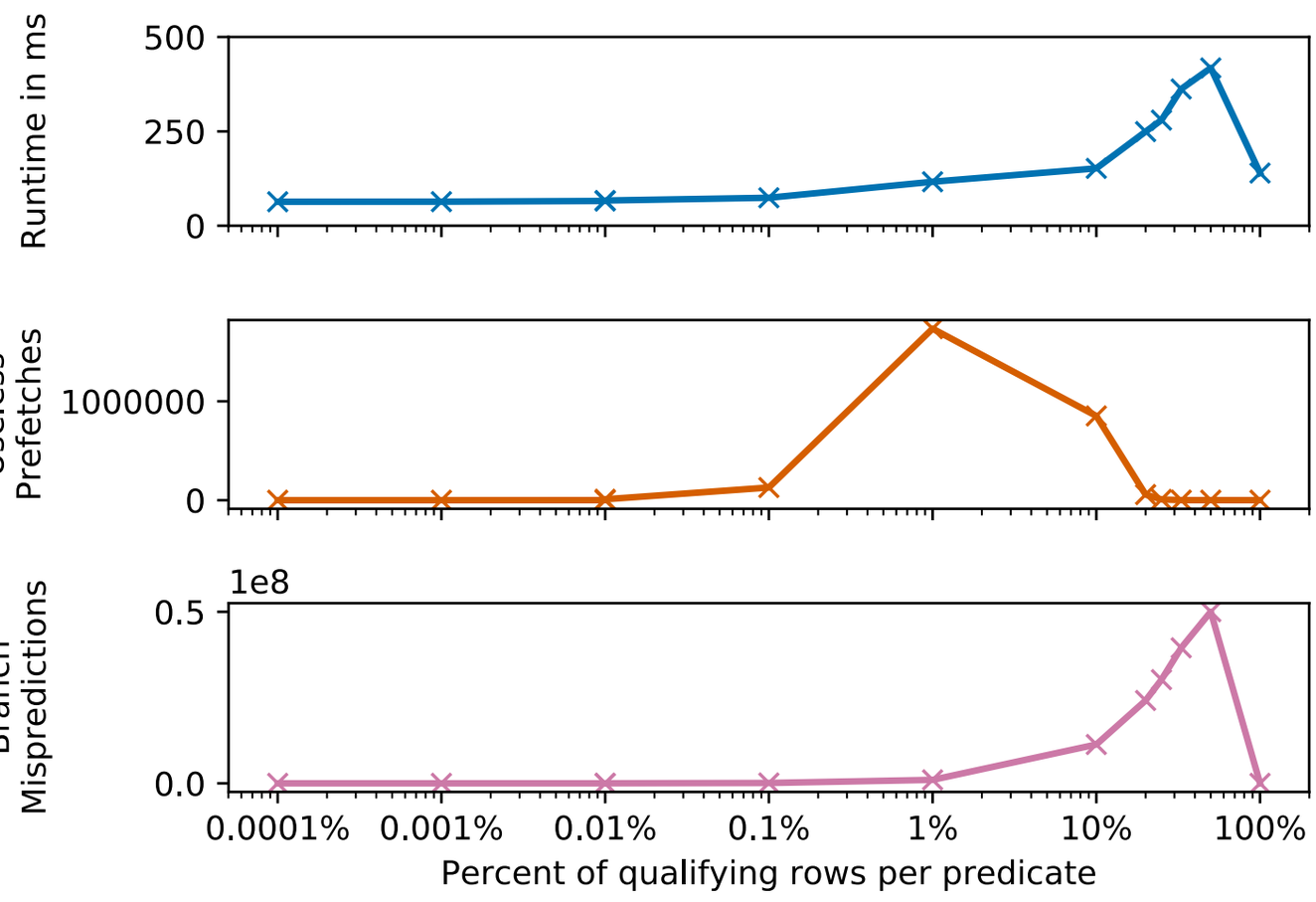
WHY SHOULD WE COMBINE DATA-CENTRIC OPERATION & SIMD?

- **Assumptions:** Data resides in-memory in column-major format with fixed size values
- `SELECT COUNT(*) FROM tbl WHERE a = 5 AND b = 2` could look similar to:

```
int total_results = 0;
for (pos_t i = 0; i < col_a.size(); ++i) {
  if (col_a[i] == 5 && col_b[i] == 2) {
    ++total_results;
  }
}
```

WHY SHOULD WE COMBINE DATA-CENTRIC OPERATION & SIMD?

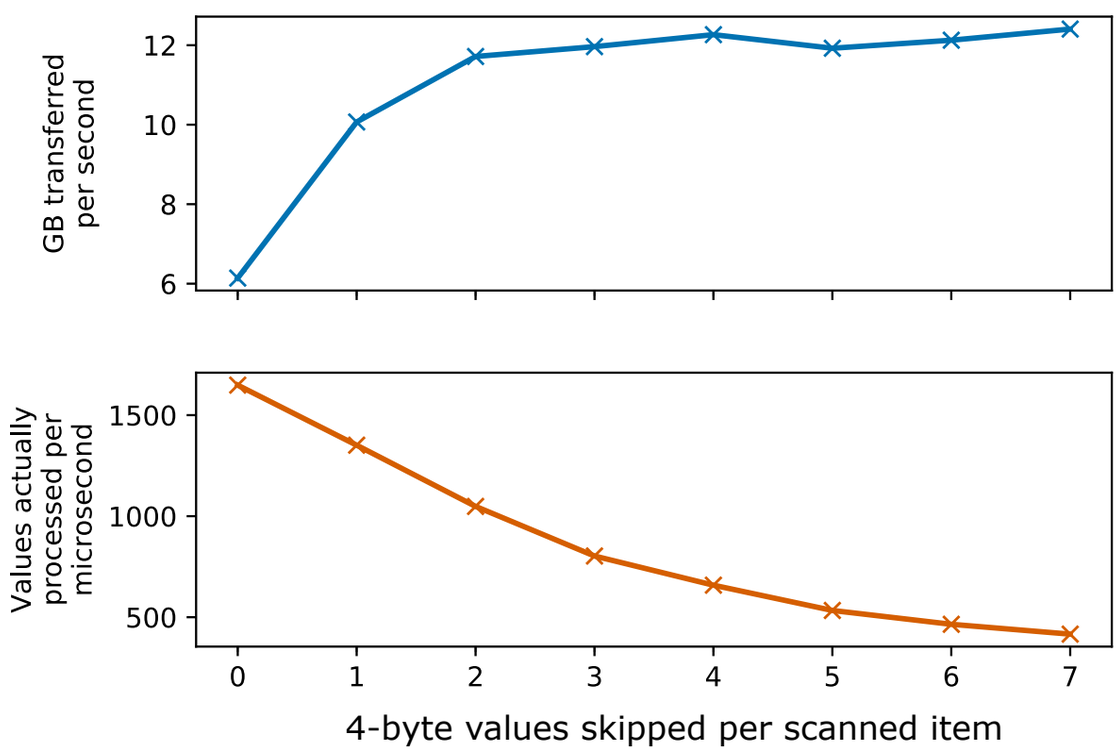
```
int total_results = 0;
for (pos_t i = 0; i < col_a.size(); ++i)
{
  if (col_a[i] == 5 && col_b[i] == 2) {
    ++total_results;
  }
}
```



HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

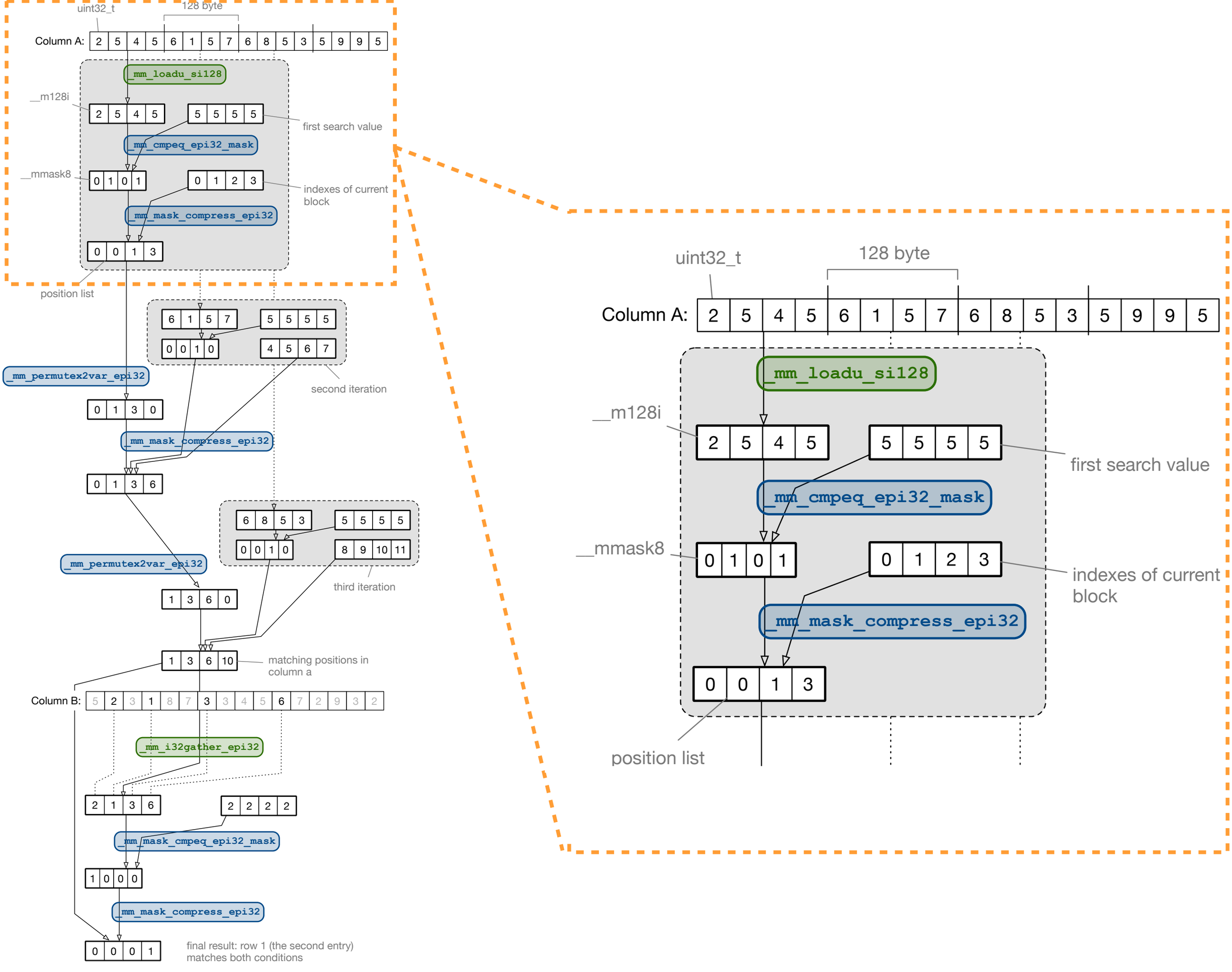
WHY SHOULD WE COMBINE DATA-CENTRIC OPERATION & SIMD?

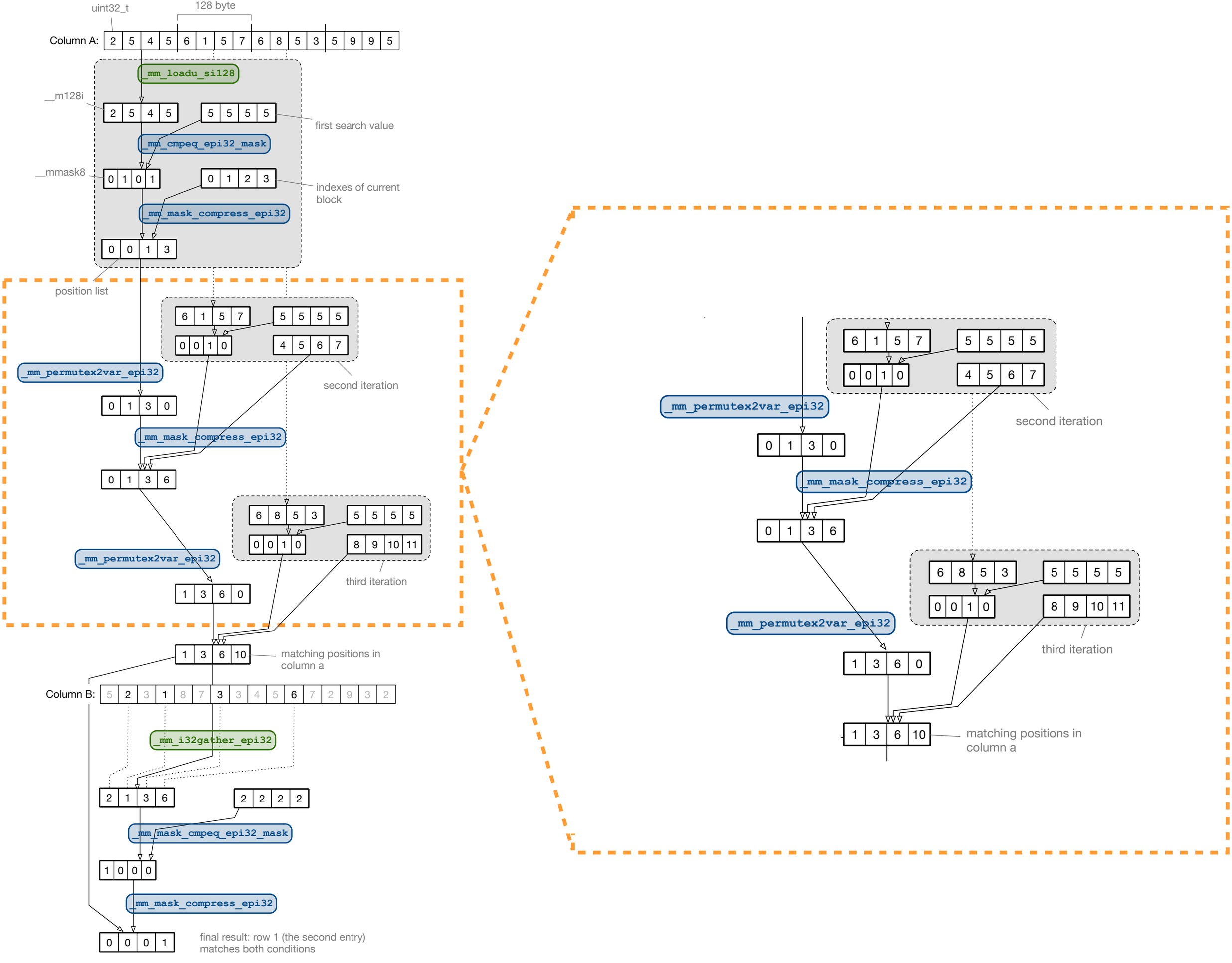
- Experiment: Does a single value at a time evaluation fully utilize the available memory bandwidth?
  - Reduce the number of cpu operations, but still load all data from memory

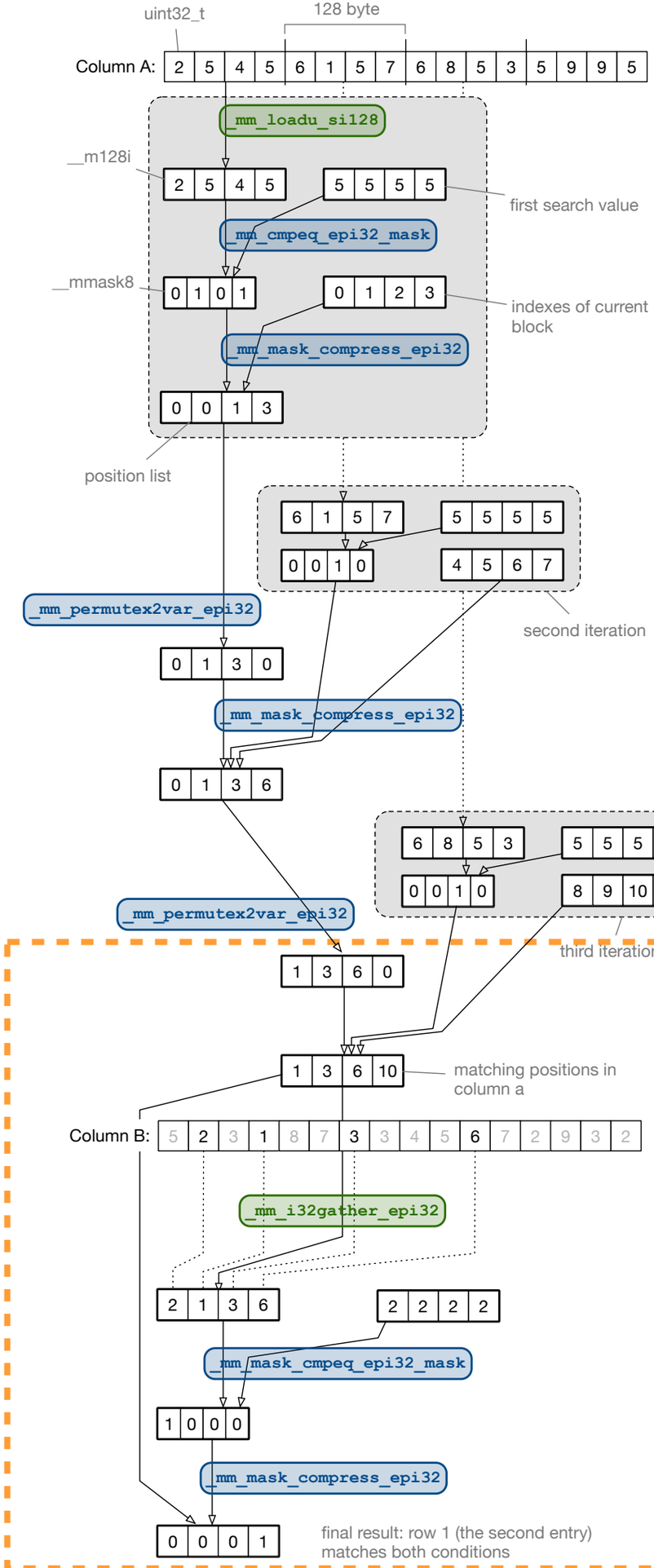


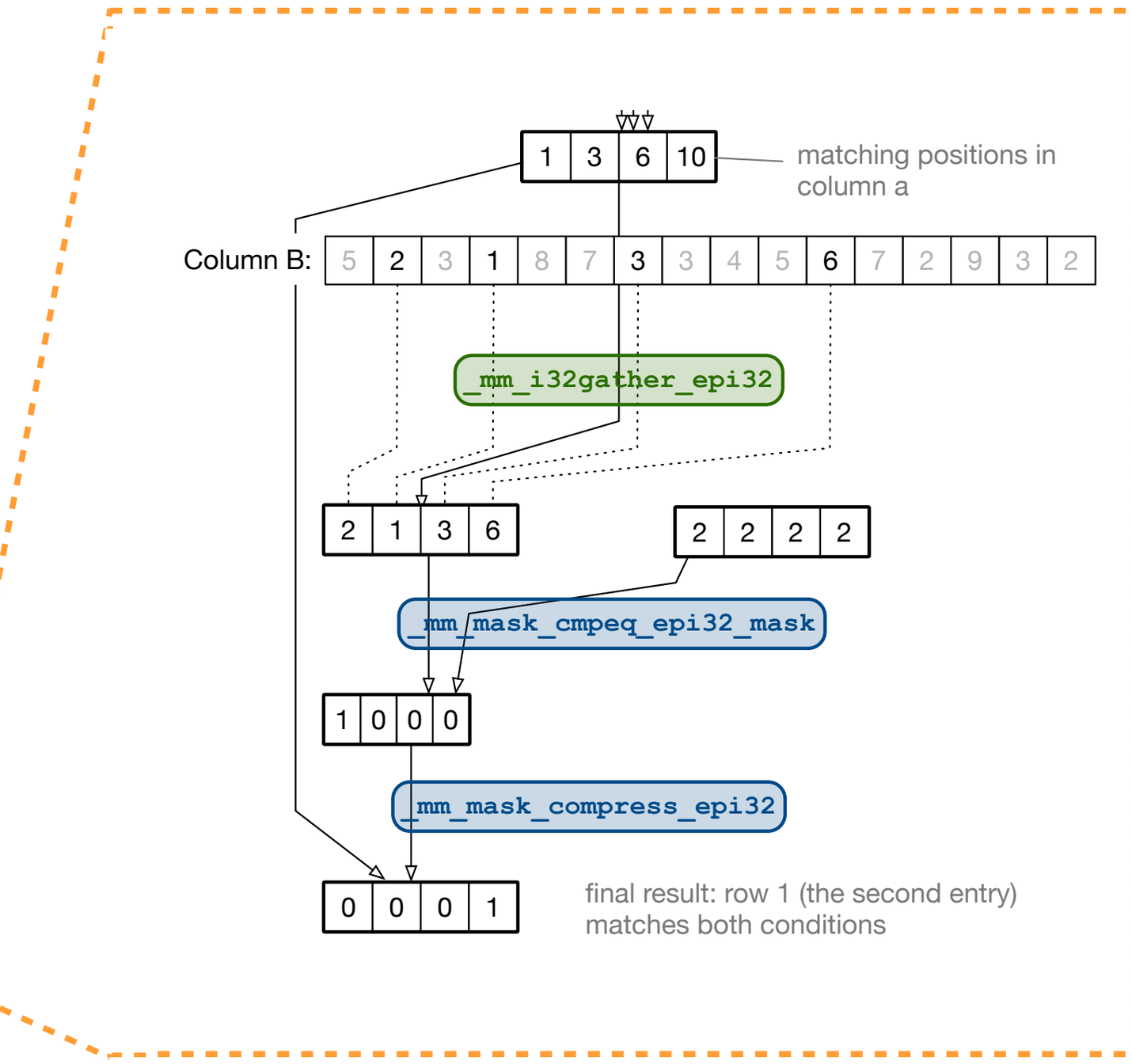
IMPLEMENTATION

- Utilizing the new instruction set AVX-512
  - Wider (doubled) register sizes
  - New instructions offer efficiency advantages
    - We built equivalent functions using AVX2 (up to 32 lines)
- **Basic idea**
  - Keep data in the AVX-registers during whole scan


uint32_t
128 byte
Column A:
2 5 4 5 6 1 5 7 6 8 5 3 5 9 9 5
_mm_loadu_si128
__m128i
2 5 4 5
5 5 5 5
first search value
_mm_cmpeq_epi32_mask
__mmask8
0 1 0 1
0 1 2 3
indexes of current block
_mm_mask_compress_epi32
0 0 1 3
position list
6 1 5 7
5 5 5 5
0 0 1 0
4 5 6 7
second iteration
_mm_permutex2var_epi32
0 1 3 0
_mm_mask_compress_epi32
0 1 3 6
6 8 5 3
5 5 5 5
0 0 1 0
8 9 10 11
third iteration
_mm_permutex2var_epi32
1 3 6 0
1 3 6 10
matching positions in column a
Column B:
5 2 3 1 8 7 3 3 4 5 6 7 2 9 3 2
_mm_i32gather_epi32
2 1 3 6
2 2 2 2
_mm_mask_cmpeq_epi32_mask
1 0 0 0
_mm_mask_compress_epi32
0 0 0 1
final result: row 1 (the second entry) matches both conditions


uint32_t
128 byte
Column A:
2 5 4 5 6 1 5 7 6 8 5 3 5 9 9 5
_mm_loadu_si128
__m128i
2 5 4 5
5 5 5 5
first search value
_mm_cmpeq_epi32_mask
__mmask8
0 1 0 1
0 1 2 3
indexes of current block
_mm_mask_compress_epi32
0 0 1 3
position list
6 1 5 7
5 5 5 5
0 0 1 0
4 5 6 7
second iteration
_mm_permutex2var_epi32
0 1 3 0
_mm_mask_compress_epi32
0 1 3 6
6 8 5 3
5 5 5 5
0 0 1 0
8 9 10 11
third iteration
_mm_permutex2var_epi32
1 3 6 0
1 3 6 10
matching positions in column a
Column B:
5 2 3 1 8 7 3 3 4 5 6 7 2 9 3 2
_mm_i32gather_epi32
2 1 3 6
2 2 2 2
_mm_mask_cmpeq_epi32_mask
1 0 0 0
_mm_mask_compress_epi32
0 0 0 1
final result: row 1 (the second entry) matches both conditions
6 1 5 7
5 5 5 5
0 0 1 0
4 5 6 7
second iteration
_mm_permutex2var_epi32
0 1 3 0
_mm_mask_compress_epi32
0 1 3 6
6 8 5 3
5 5 5 5
0 0 1 0
8 9 10 11
third iteration
_mm_permutex2var_epi32
1 3 6 0
1 3 6 10
matching positions in column a

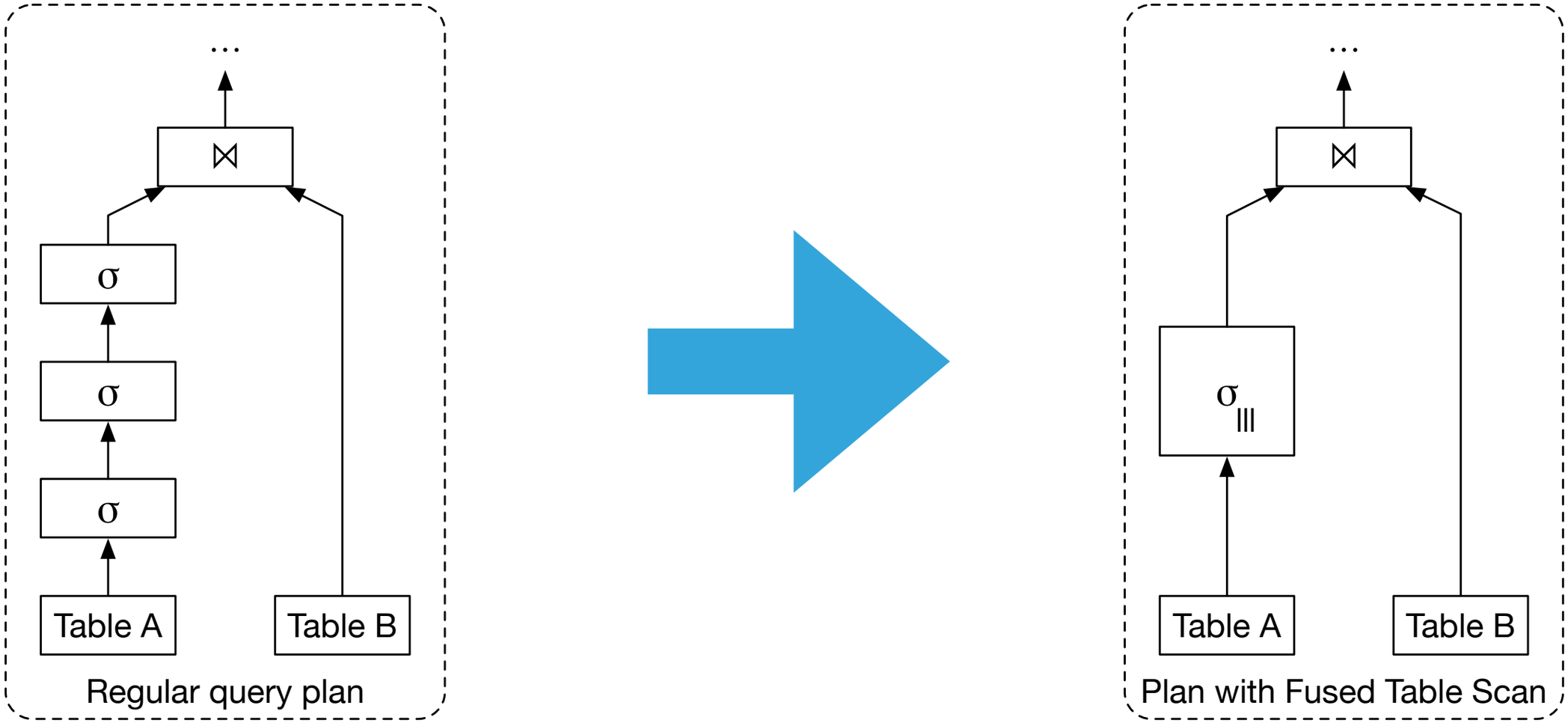

uint32_t
128 byte
Column A:
2 5 4 5 6 1 5 7 6 8 5 3 5 9 9 5
_mm_loadu_si128
__m128i
2 5 4 5
5 5 5 5
first search value
_mm_cmpeq_epi32_mask
__mmask8
0 1 0 1
0 1 2 3
indexes of current block
_mm_mask_compress_epi32
0 0 1 3
position list
6 1 5 7
5 5 5 5
0 0 1 0
4 5 6 7
second iteration
_mm_permutex2var_epi32
0 1 3 0
_mm_mask_compress_epi32
0 1 3 6
6 8 5 3
5 5 5 5
0 0 1 0
8 9 10 11
third iteration
_mm_permutex2var_epi32
1 3 6 0
1 3 6 10
matching positions in column a
Column B:
5 2 3 1 8 7 3 3 4 5 6 7 2 9 3 2
_mm_i32gather_epi32
2 1 3 6
2 2 2 2
_mm_mask_cmpeq_epi32_mask
1 0 0 0
_mm_mask_compress_epi32
0 0 0 1
final result: row 1 (the second entry) matches both conditions


1 3 6 10
matching positions in column a
Column B:
5 2 3 1 8 7 3 3 4 5 6 7 2 9 3 2
_mm_i32gather_epi32
2 1 3 6
2 2 2 2
_mm_mask_cmpeq_epi32_mask
1 0 0 0
_mm_mask_compress_epi32
0 0 0 1
final result: row 1 (the second entry) matches both conditions

IMPLEMENTATION – ACHIEVEMENTS

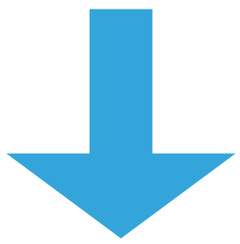
- Fully utilize the CPU's computation power
  - More comparisons/cycle by using SIMD instructions
- Avoid useless prefetches, only load necessary data of the second column
  - Increased memory bus efficiency
- Fewer and cheaper branch mispredictions
  - Reduced number of conditions in code
- Reduced memory transfers
  - Intermediary results are kept in AVX registers and are not materialized

JIT – RUNTIME CODE GENERATION



JIT – RUNTIME CODE GENERATION

- **Problem:** Some parameters are only known at runtime
  - Size of scanned values
  - Exact data types
    - Signed & unsigned 1, 2, 4, or 8 byte int plus float & double
  - Type of comparison operators: !=, ==, <, >, <=, >=

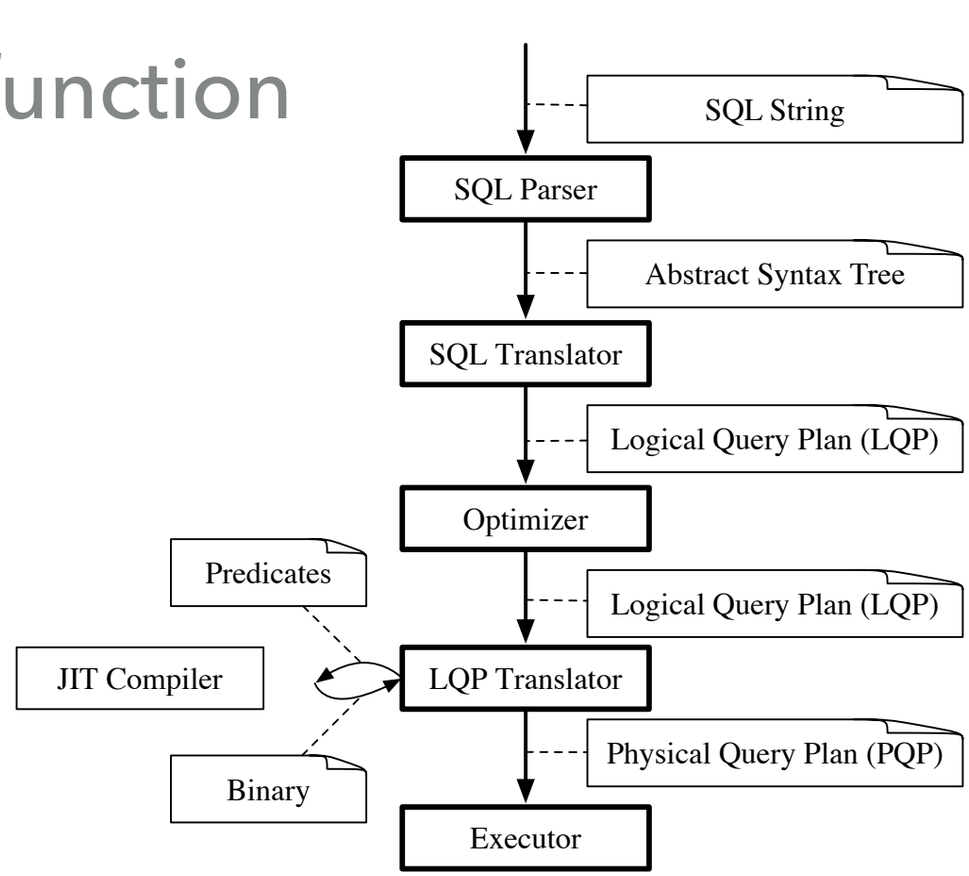


- Larger number of possible code paths

JIT – RUNTIME CODE GENERATION

- The query *optimizer* identifies fusable operator chains
- Parameters are determined by the *translator* during runtime
- **Result:** Specialized, monolithic function
- Cached for efficiency

https://github.com/hyrise/hyrise/

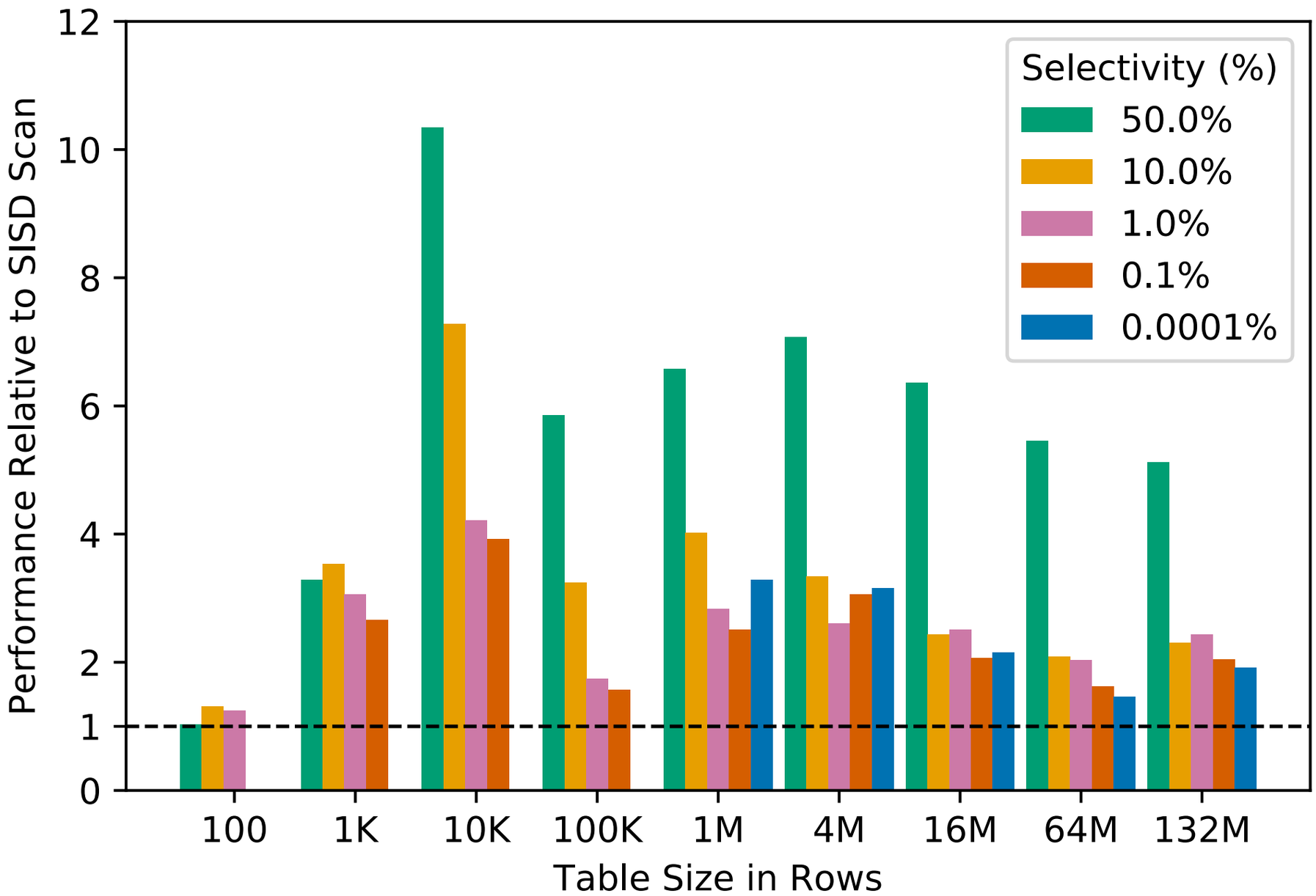


EVALUATION

- On current Skylake system
  - Intel Xeon Platinum @ 2.5 - 3.8 GHz with 2TB of PC4-2666 main memory
- Evaluated dimensions during experiments
  - Table Size
  - Selectivity
  - Implementations / Instruction sets
    - SISD, AVX2, and AVX-512, automatic compiler vectorization
  - AVX-Register width: 128, 256, and 512 Bit
  - Number of Predicates

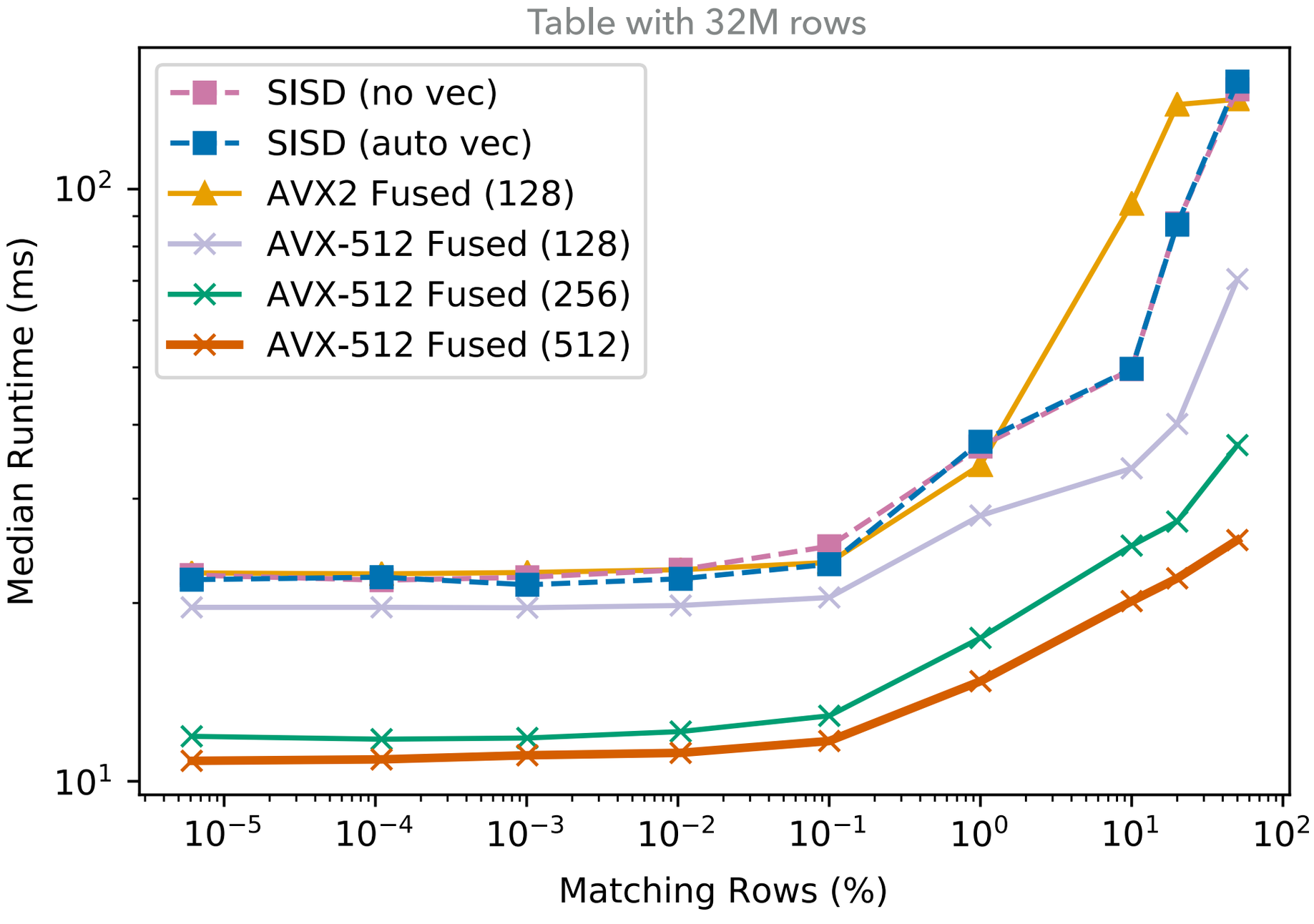
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

EVALUATION – PERFORMANCE RELATIVE TO SISD IMPLEMENTATION



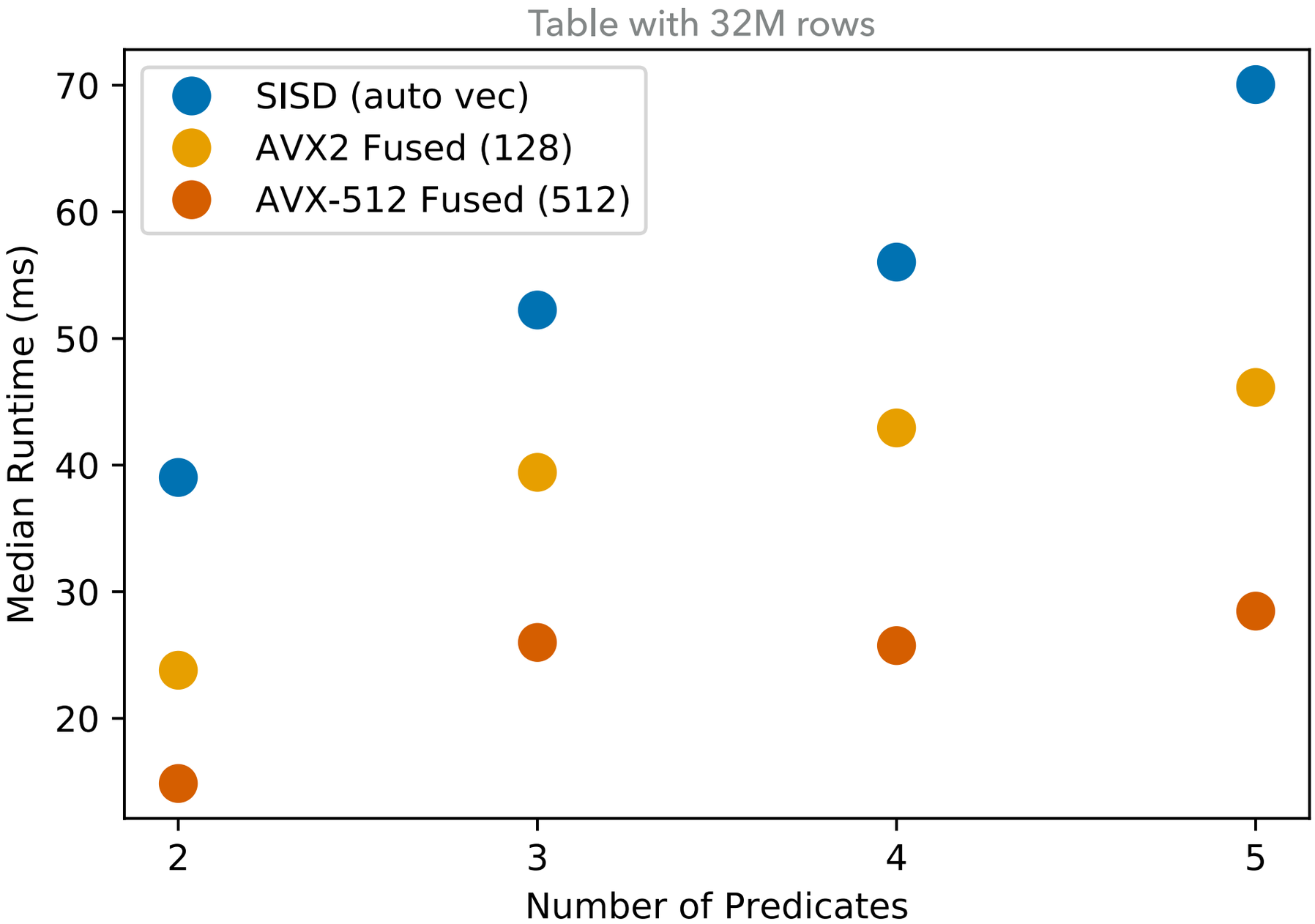
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

EVALUATION – INSTRUCTION SETS & REGISTER WIDTH



HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

EVALUATION – NUMBER OF PREDICATES



SUMMARY & CONCLUSION

- Branch mispredictions and useless prefetches are a huge cost factor in multi-predicate scans
- Doubling the register size does not (yet) double the performance
- Bringing together AVX-512 with Just-In-Time compilation
  - Use new AVX-512 instructions to efficiently load and remove tuples from AVX-registers without leaving SIMD mode
  - Performance was at least doubled in 80% of test cases
- Future Work: Impact of other encoding methods