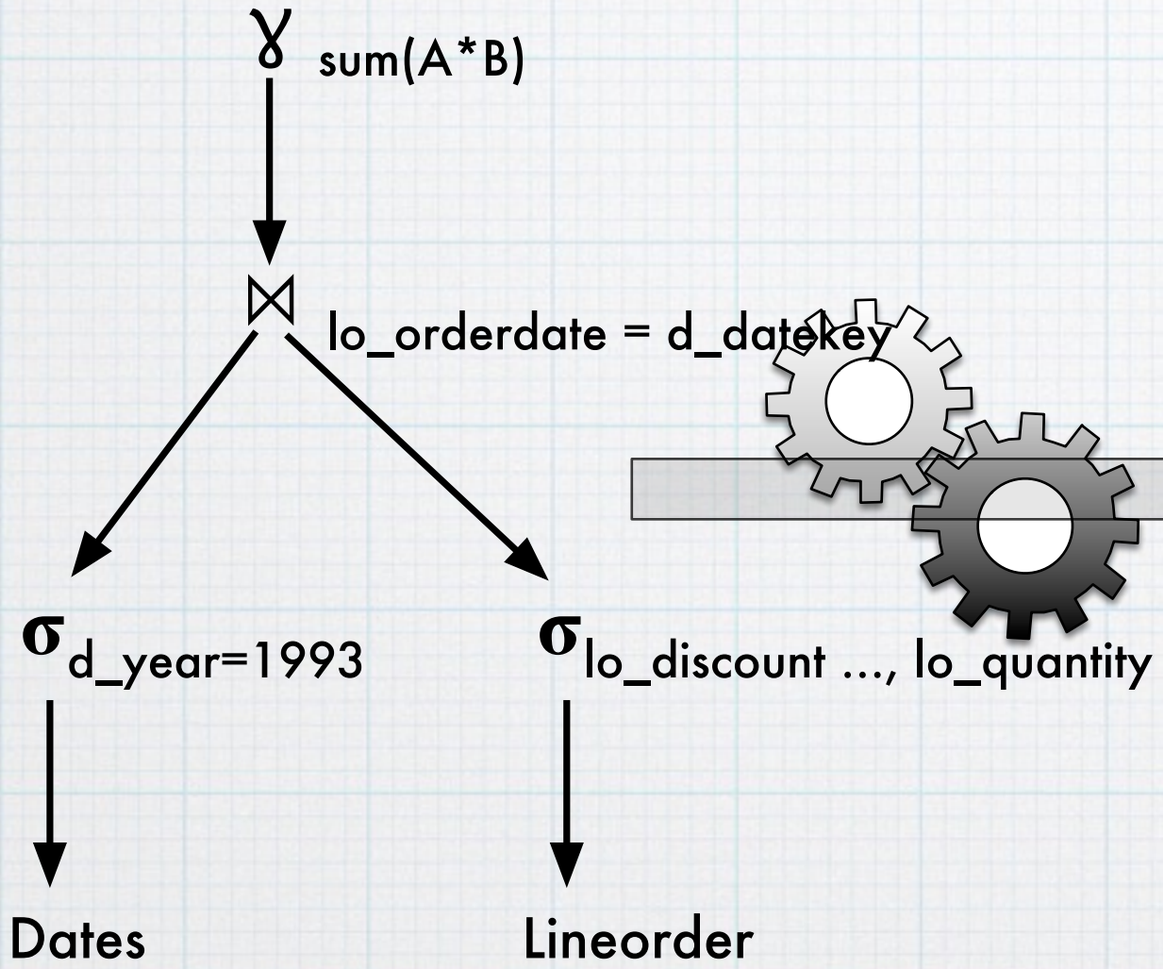
Hardware-Sensitive Scan Operator Variants for Compiled Selection Pipelines

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Databases and Software Engineering

David Broneske, Andreas Meister, Gunter Saake
University of Magdeburg

Introduction Query Compilation

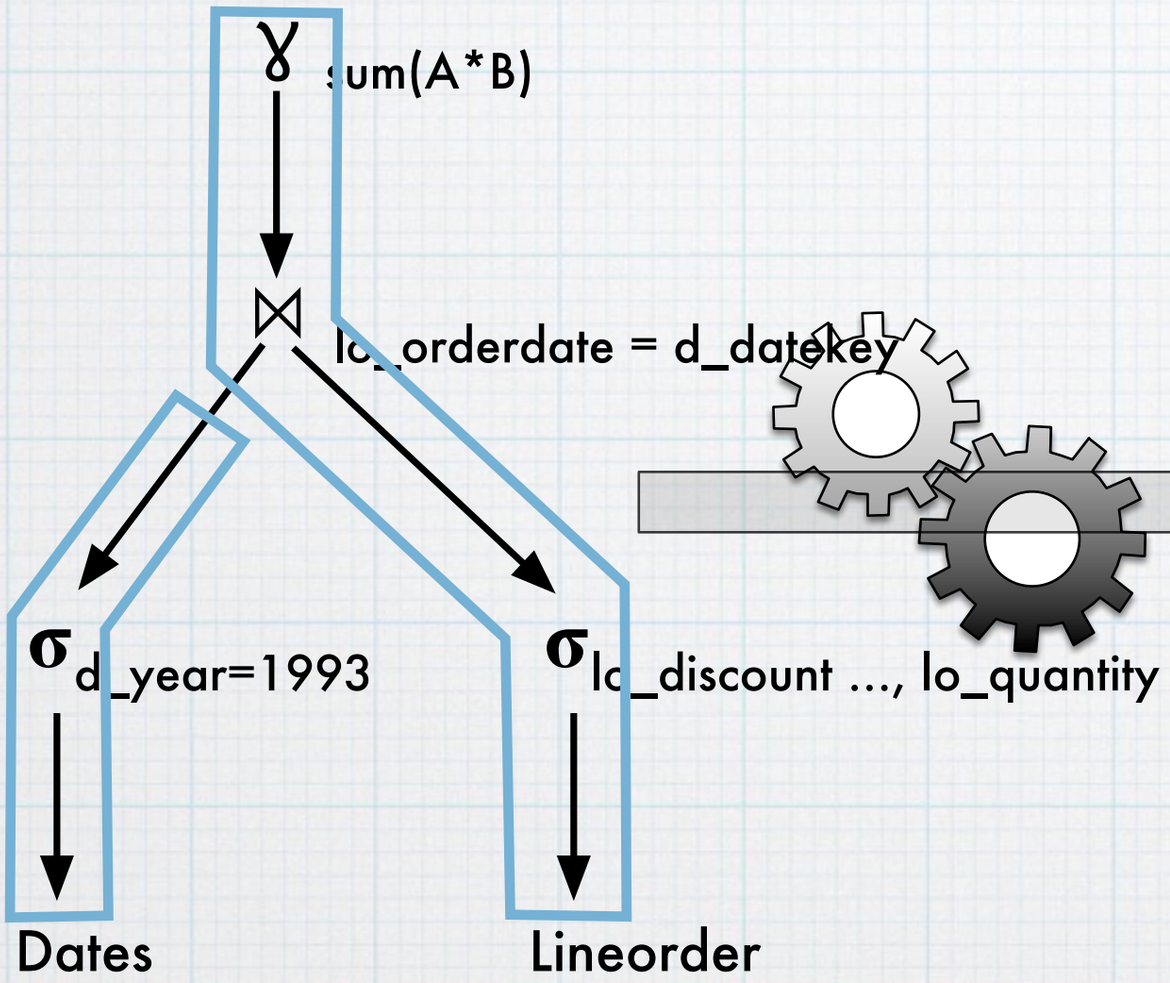



```
int32_t* array_LINEORDER1_LINEORDER_LO_QUANTITY1 = getArrayFromColumn_int32_t(col_LINEORDER1_LINEORDER_LO_QUANTITY1);

TID tuple_id_DATES1=0;
HashTablePtr generic_hashtable_DATES_D_DATEKEY1=getHashTable(table_DATES1, "DATES.D_DATEKEY.1");
hashtable_t* hashtable_DATES_D_DATEKEY1= (hashtable_t*) getHashTableFromSystemHashTable(generic_hashtable_DATES_D_DATEKEY1);
double computed_var = double(0);
SUM_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM = 0;


size_t current_result_size=0;
size_t allocated_result_elements=10000;
double* result_array_REVENUE = (double*) realloc(NULL, allocated_result_elements * sizeof(double));

size_t num_elements_LINEORDER_1 = getNumberOfRows(table_LINEORDER1);
for(size_t tuple_id_LINEORDER1=0; tuple_id_LINEORDER1<num_elements_LINEORDER_1; ++tuple_id_LINEORDER1) {
    if((( array_LO_DISCOUNT1[tuple_id_LINEORDER1] >= 1.0f))) {
        if((( array_LO_DISCOUNT1[tuple_id_LINEORDER1] <= 3.0f))) {
            if(((array_LINEORDER1_LINEORDER_LO_QUANTITY1[tuple_id_LINEORDER1] < 25))) {
                unsigned long hash_DATES_D_DATEKEY1 = HASH(array_LO_ORDERDATE1[tuple_id_LINEORDER1]) & hashtable_DATES_D_DATEKEY1->mask;
                hash_bucket_t *b_Datek = &hashtable_DATES_D_DATEKEY1->buckets[hash_DATES_D_DATEKEY1];
                while (b_Datek) {
                    for (size_t b_tid_Datek = 0; b_tid_Datek < b_Datek->count; b_tid_Datek++) {
                        if (b_Datek->tuples[b_tid_Datek].key == array_LO_ORDERDATE1[tuple_id_LINEORDER1]) {
                            tuple_id_DATES1=b_Datek->tuples[b_tid_Datek].value;

                            computed_var = array_LO_EXTENDEDPRICE1[tuple_id_LINEORDER1] *  array_LO_DISCOUNT1[tuple_id_LINEORDER1];
                            SUM += computed_var;
                        }
                    }
                    b_Datek = b_Datek->next;
                }

            }
        }
    }
}
result_array_REVENUE[current_result_size] = SUMarray_LO_ORDERDATE1_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM;
current_result_size++;
std::vector<ColumnPtr> result_columns;
result_columns.push_back(createResultArray_double("REVENUE", result_array_REVENUE, current_result_size));
TablePtr result_table=createTableFromColumns("LINEORDER", &result_columns[0], result_columns.size());
/* Add build hash tables to result table. */

/* Clean up resources. */

return result_table;
```

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

DBSE

Introduction Query Compilation

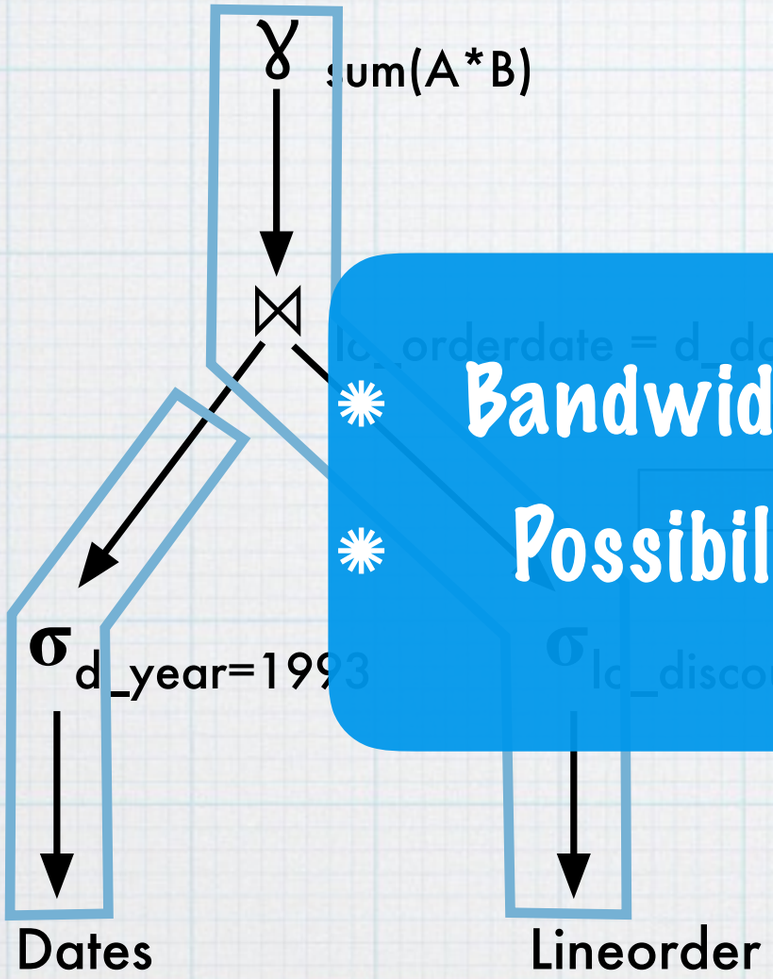



```
int32_t* array_LINEORDER1_LINEORDER_LO_QUANTITY1 = getArrayFromColumn_int32_t(col_LINEORDER1_LINEORDER_LO_QUANTITY1);

TID tuple_id_DATES1=0;
HashTablePtr generic_hashtable_DATES_D_DATEKEY1=getHashTable(table_DATES1, "DATES.D_DATEKEY.1");
hashtable_t* hashtable_DATES_D_DATEKEY1= (hashtable_t*) getHashTableFromSystemHashTable(generic_hashtable_DATES_D_DATEKEY1);
double computed_var = double(0);
SUM_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM = 0;


size_t current_result_size=0;
size_t allocated_result_elements=10000;
double* result_array_REVENUE = (double*) realloc(NULL, allocated_result_elements * sizeof(double));

size_t num_elements_LINEORDER_1 = getNumberOfRows(table_LINEORDER1);
for(size_t tuple_id_LINEORDER1=0; tuple_id_LINEORDER1<num_elements_LINEORDER_1; ++tuple_id_LINEORDER1) {
    if((( array_LO_DISCOUNT1[tuple_id_LINEORDER1] >= 1.0f))) {
        if(((  array_LO_DISCOUNT1[tuple_id_LINEORDER1] <= 3.0f))) {
            if(((array_LINEORDER1_LINEORDER_LO_QUANTITY1[tuple_id_LINEORDER1] < 25))) {
                unsigned long hash_DATES_D_DATEKEY1 = HASH(array_LO_ORDERDATE1[tuple_id_LINEORDER1]) & hashtable_DATES_D_DATEKEY1->mask;
                hash_bucket_t *b_Datek = &hashtable_DATES_D_DATEKEY1->buckets[hash_DATES_D_DATEKEY1];
                while (b_Datek) {
                    for (size_t b_tid_Datek = 0; b_tid_Datek < b_Datek->count; b_tid_Datek++) {
                        if (b_Datek->tuples[b_tid_Datek].key == array_LO_ORDERDATE1[tuple_id_LINEORDER1]) {
                            tuple_id_DATES1=b_Datek->tuples[b_tid_Datek].value;

                            computed_var = array_LO_EXTENDEDPRICE1[tuple_id_LINEORDER1] *  array_LO_DISCOUNT1[tuple_id_LINEORDER1];
                            SUM += computed_var;
                        }
                    }
                    b_Datek = b_Datek->next;
                }

            }
        }
    }
}
result_array_REVENUE[current_result_size] = SUMarray_LO_ORDERDATE1_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM;
current_result_size++;
std::vector<ColumnPtr> result_columns;
result_columns.push_back(createResultArray_double("REVENUE", result_array_REVENUE, current_result_size));
TablePtr result_table=createTableFromColumns("LINEORDER", &result_columns[0], result_columns.size());
/* Add build hash tables to result table. */

/* Clean up resources. */

return result_table;
```

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

D B S E

Introduction Query Compilation



- Bandwidth-bound -> compute-bound
- Possibility for code optimizations

```
int32_t* array_LINEORDER1_LINEORDER_LO_QUANTITY1 = getArrayFromColumn_int32_t(col_LINEORDER1_LINEORDER_LO_QUANTITY1);

TID tuple_id_DATES1=0;
HashTablePtr generic_hashtable_DATES_D_DATEKEY1=getHashTable(table_DATES1, "DATES.D_DATEKEY.1");
hashtable_t* hashtable_DATES_D_DATEKEY1= (hashtable_t*) getHashTableFromSystemHashTable(generic_hashtable_DATES_D_DATEKEY1);
double computed_var = double(0);
SUM_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM = 0;
```

```
result_array_REVENUE[current_result_size] = SUMarray_LO_ORDERDATE1_OF_LINEORDER_LO_EXTENDEDPRICE1_MUL_LINEORDER_LO_DISCOUNT1_SUM;
current_result_size++;
std::vector<ColumnPtr> result_columns;
result_columns.push_back(createFixedSizeDoubleColumn("REVENUE", result_array_REVENUE, current_result_size));
TablePtr result_table=createTableFromColumns("LINEORDER", &result_columns[0], result_columns.size());
/* Add build hash tables to result table. */

/* Clean up resources. */

return result_table;
```

Motivating Examples

Motivating Examples

Branching

```
1  for(int i = 0; i < input_size; ++i){
2    if(col[i] < pred)
3        agg+=agg_col[i];
4  }
```

Motivating Examples

Branching

```
1  for(int i = 0; i < input_size; ++i){
2    if(col[i] < pred)
3        agg+=agg_col[i];
4  }
```

Predicated

```
1  for(int i = 0; i < input_size; ++i){
2    agg+=agg_col[i]*(col[i] < pred);
3  }
```

Motivating Examples

Branching

```
1  for(int i = 0; i < input_size; ++i){
2    if(col[i] < pred)
3        agg+=agg_col[i];
4  }
```

Predicated

```
1  for(int i = 0; i < input_size; ++i){
2    agg+=agg_col[i]*(col[i] < pred);
3  }
```

SIMD [ZR02]

```
1  for(int i = 0; i < simd_size; ++i){
2    mask= SIMD_COMP(simd_col[i],pred);
3    if(mask){
4      for (int j=0;j < SIMD_LENGTH;++j){
5        if((mask >> j) & 1)
6          agg+=agg_col[i];
7      }
8    }
9  }
```

Motivating Examples

Branching

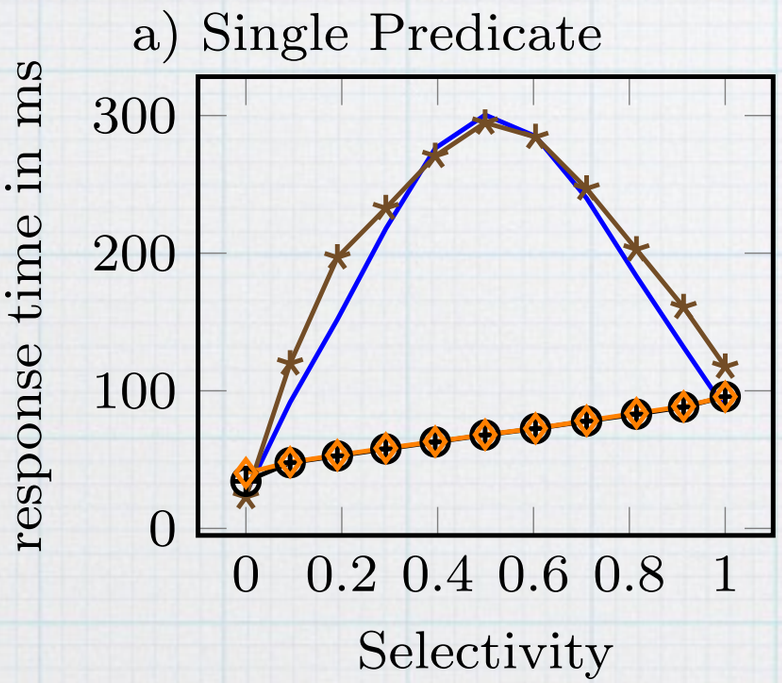
```
1  for(int i = 0; i < input_size; ++i){
2    if(col[i] < pred)
3        agg+=agg_col[i];
4  }
```

Predicated

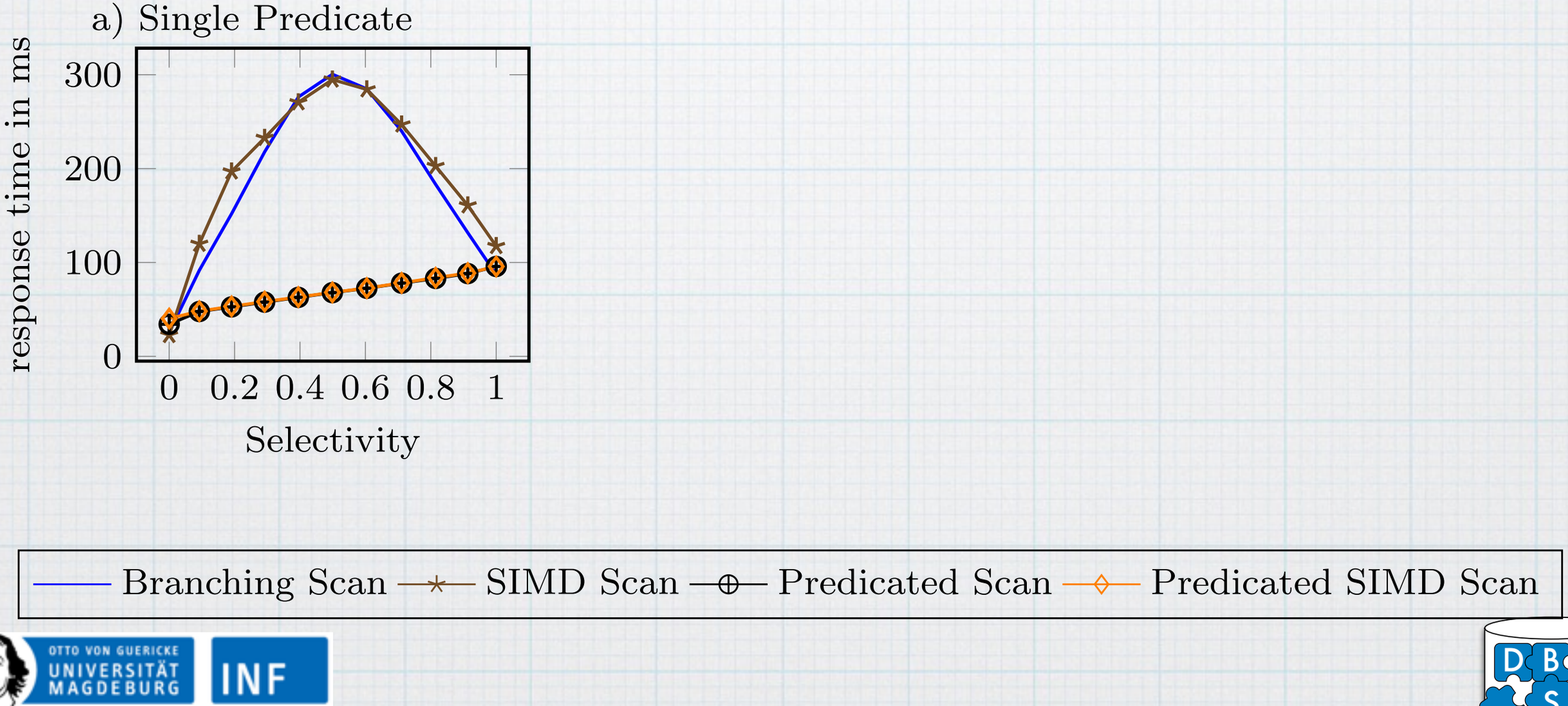
```
1  for(int i = 0; i < input_size; ++i){
2    agg+=agg_col[i]*(col[i] < pred);
3  }
```

SIMD [ZR02]

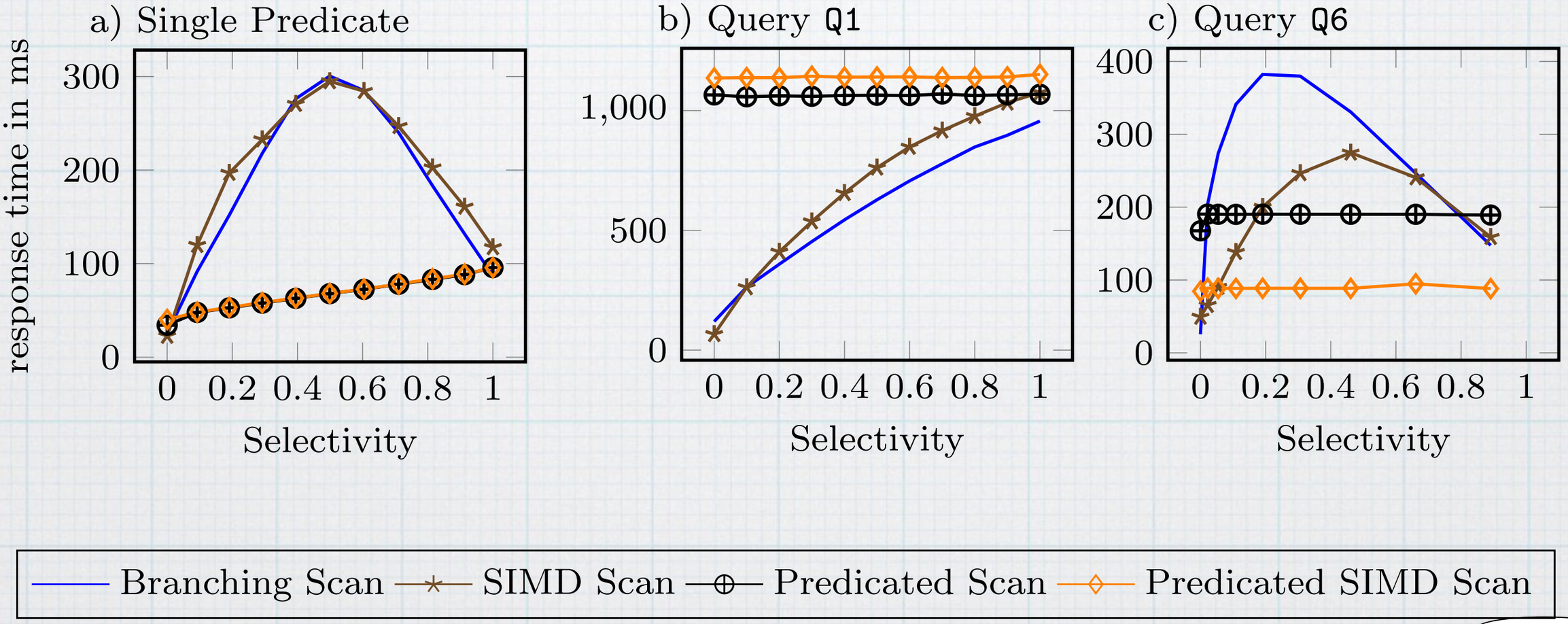
```
1  for(int i = 0; i < simd_size; ++i){
2    mask= SIMD_COMP(simd_col[i],pred);
3    if(mask){
4      for (int j=0;j < SIMD_LENGTH;++j){
5        if((mask >> j) & 1)
6          agg+=agg_col[i];
7      }
8    }
9  }
```



Motivating Examples

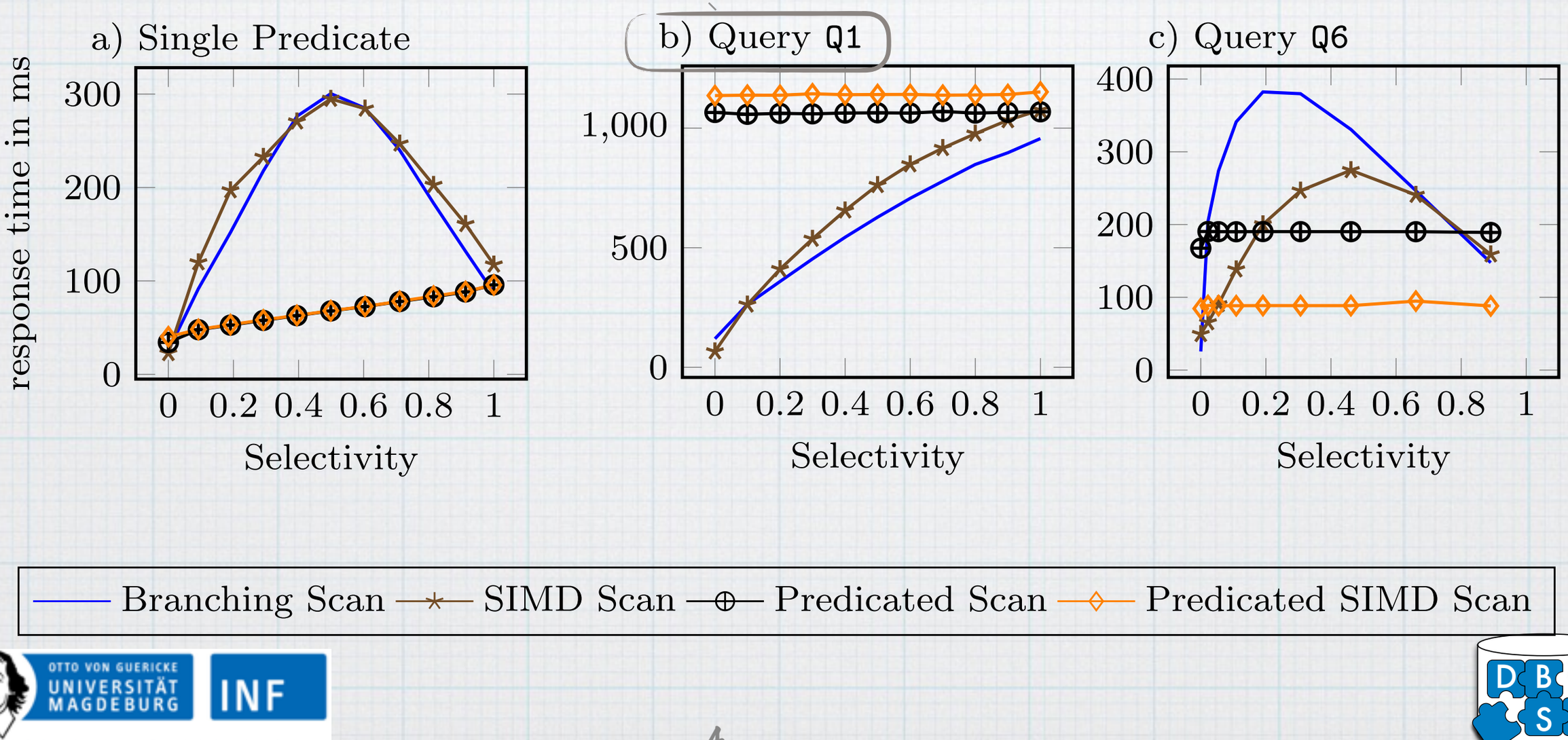


Motivating Examples



Motivating Examples

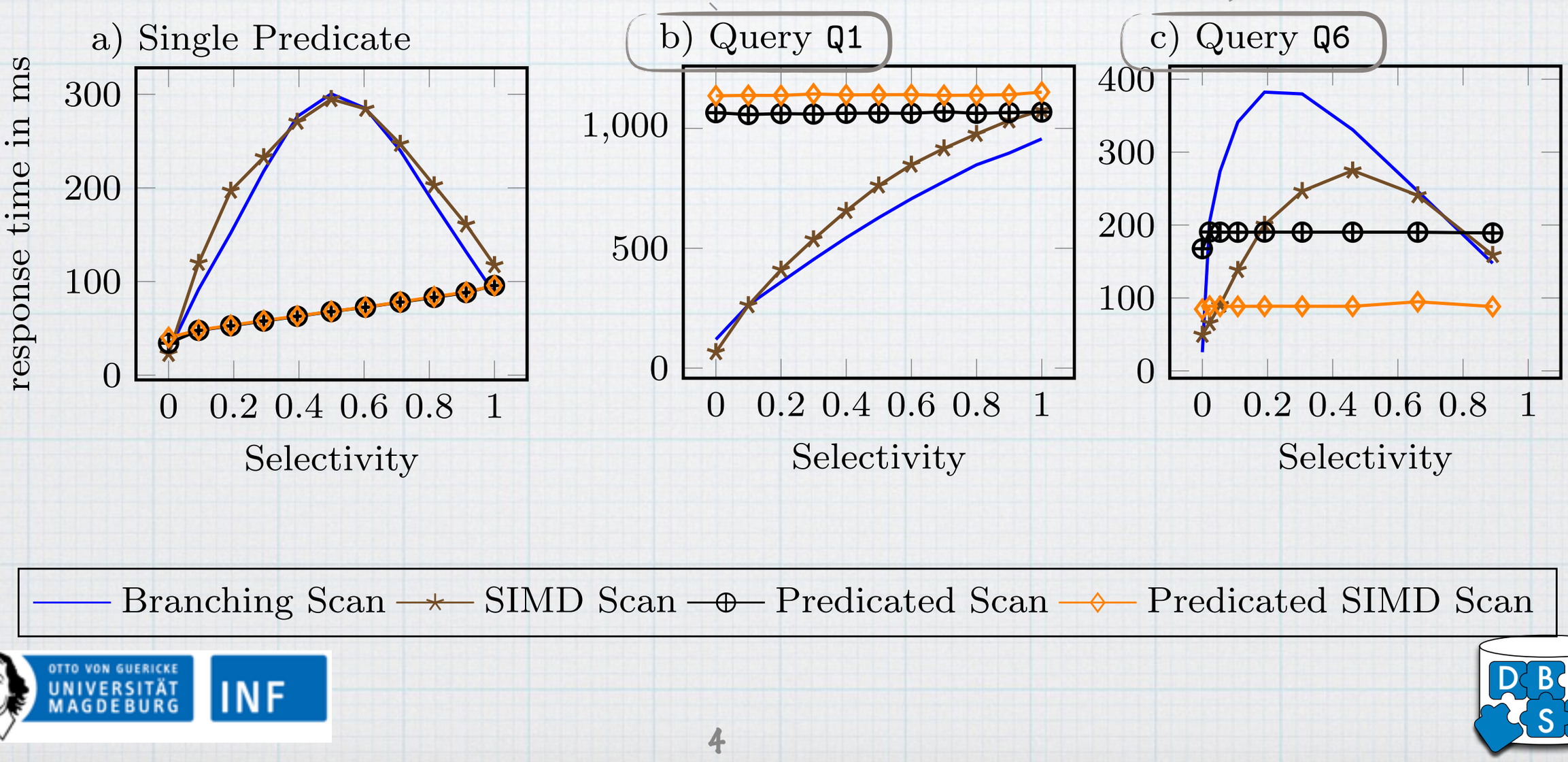
8 Aggregates
1 Filter Predicate



Motivating Examples

8 Aggregates
1 Filter Predicate

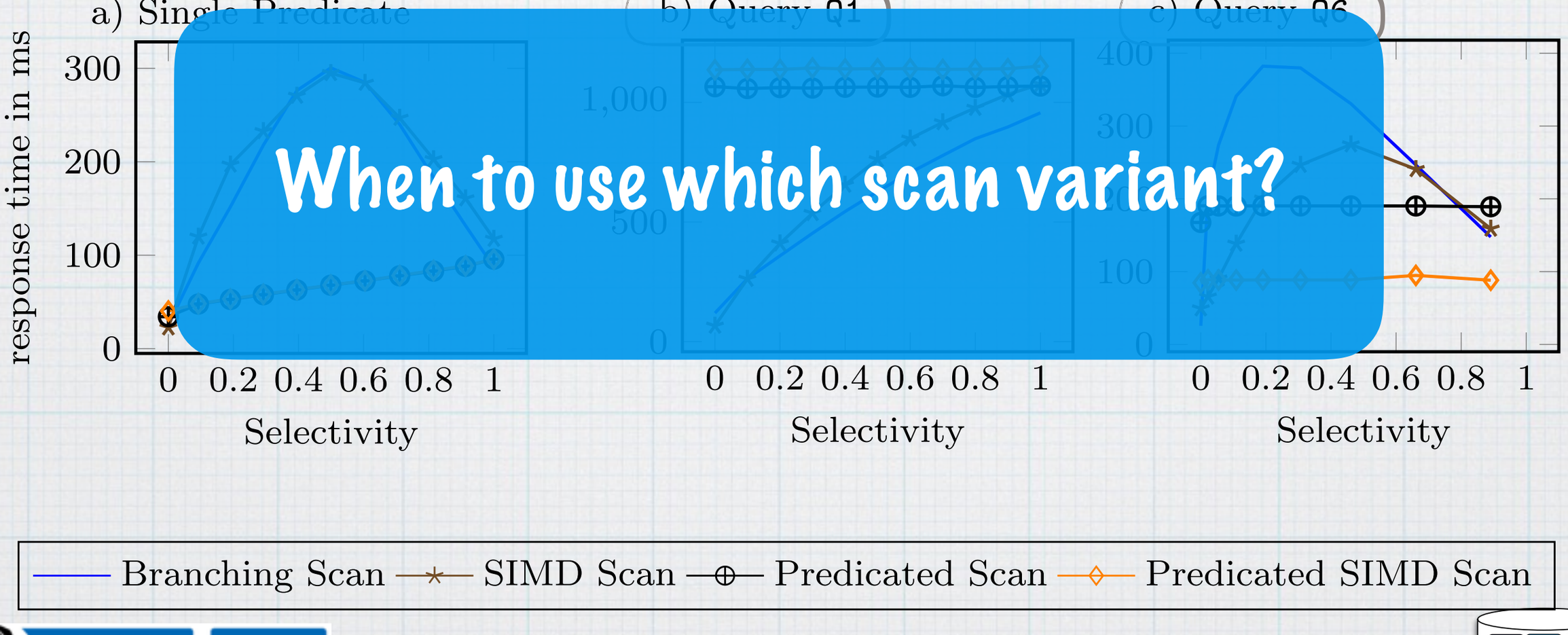
1 Aggregate
3 Filter Predicates



Motivating Examples

8 Aggregates
1 Filter Predicate

1 Aggregate
3 Filter Predicates



When to use which scan variant?

Evaluation Setup

Evaluation Criteria

* Number of predicates
* Number of aggregates inside loop

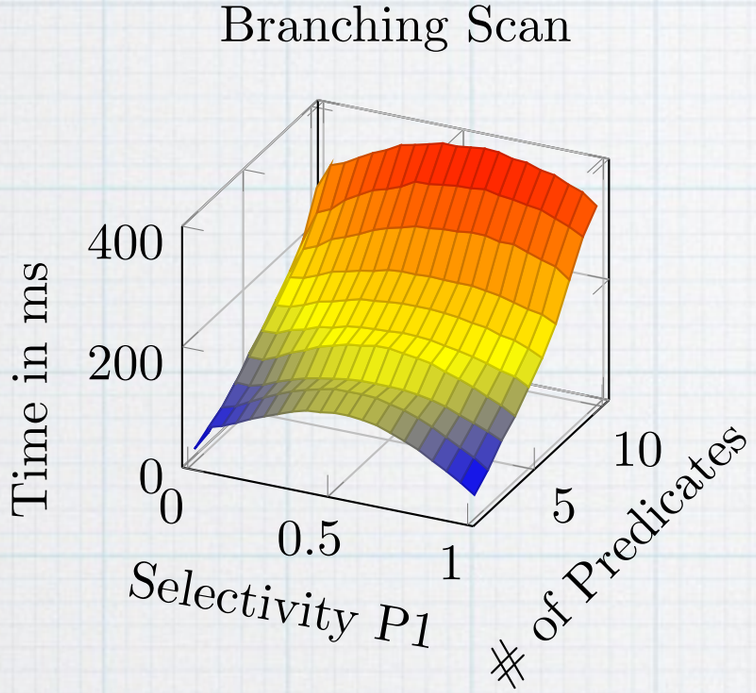
Workload & Machine

* TPC-H Lineltem table SF 10
* Intel Xeon E5-2630 v3 with SSE4.2

Variants:

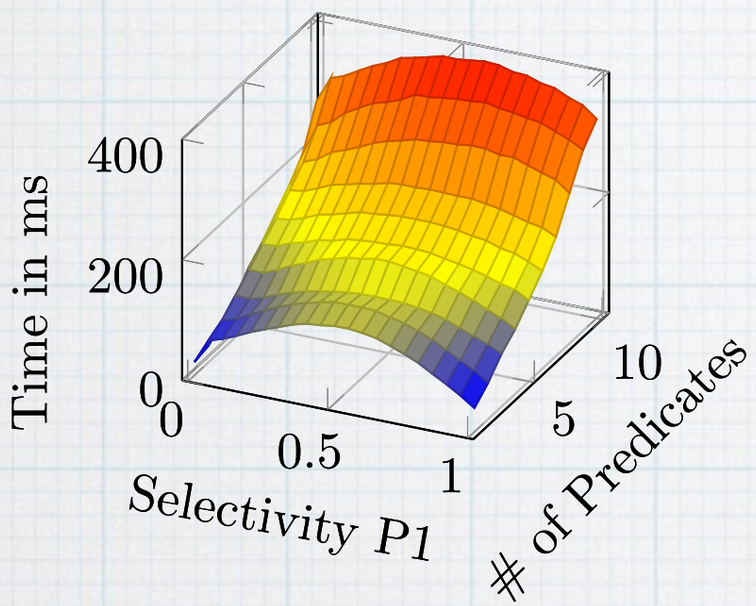
* Branching vs. Predication
* Scalar vs. SIMD

Number of Predicates

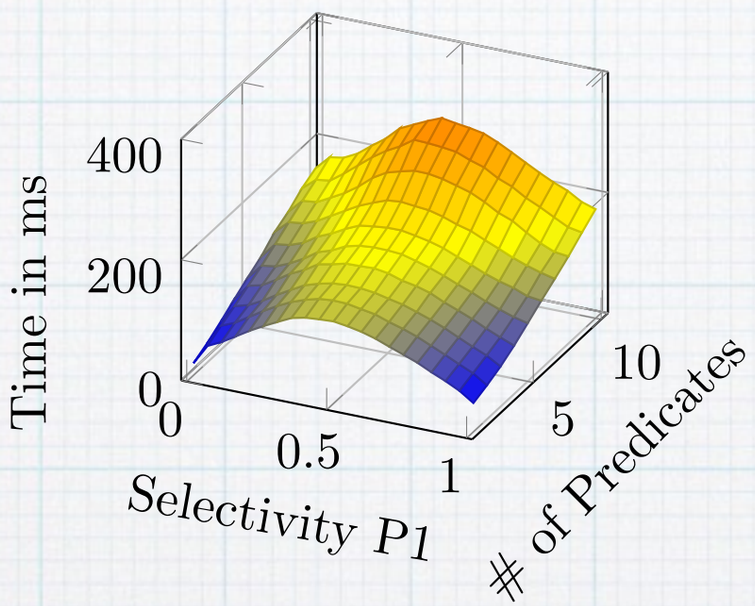


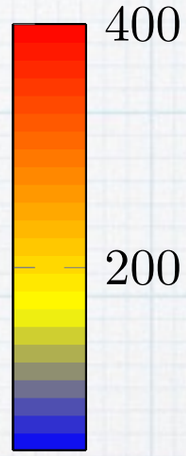
Number of Predicates

Branching Scan

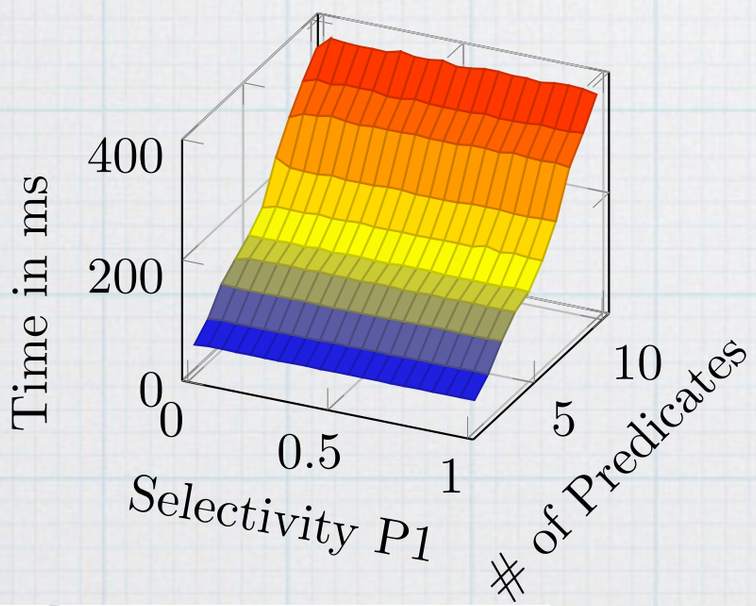


SIMD Scan

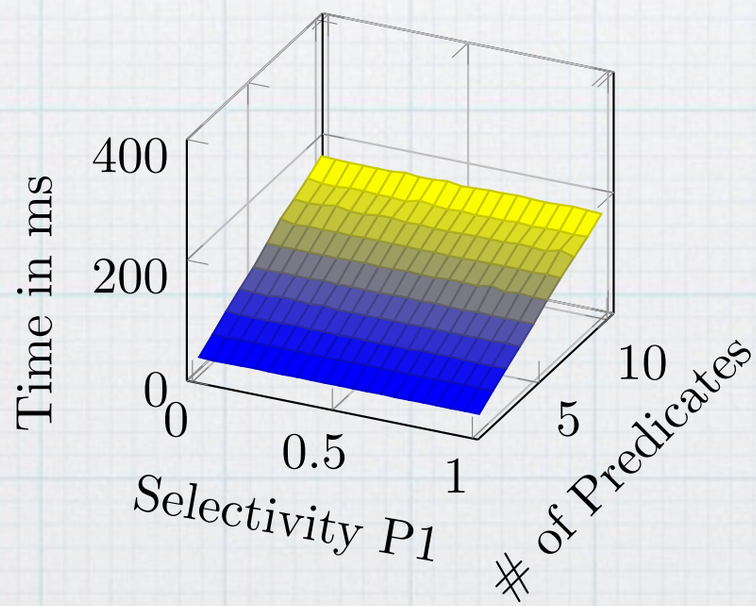




Predicated Scan



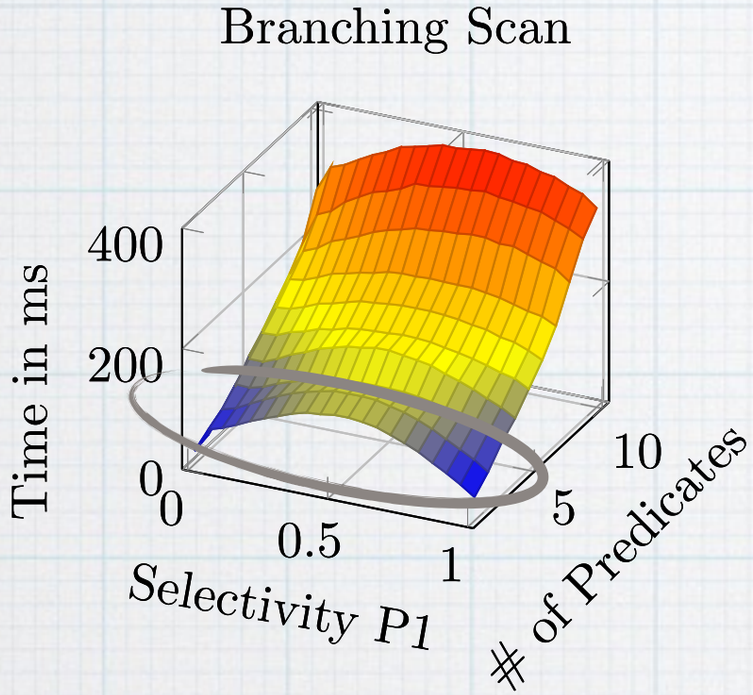
SIMD Predicated Scan

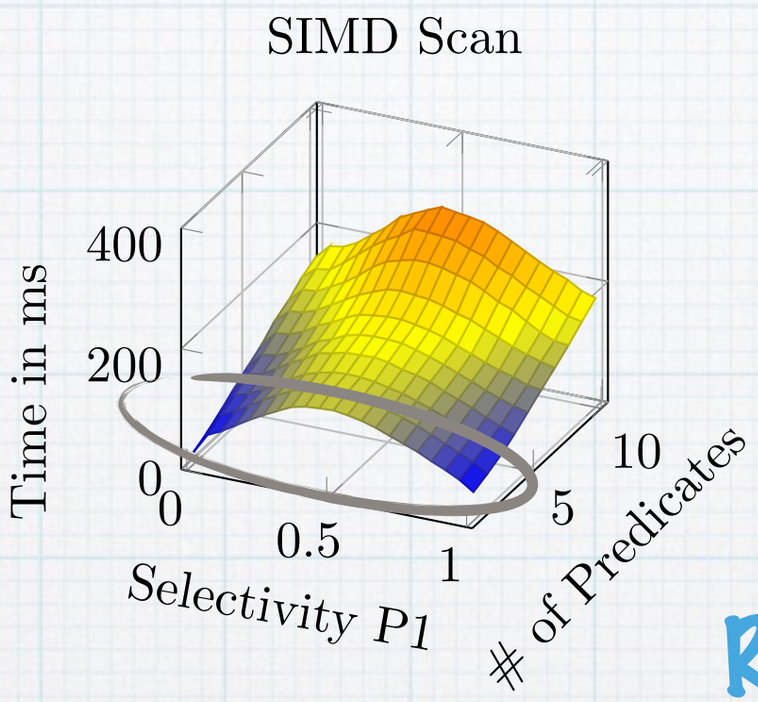


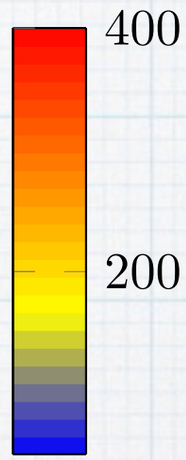
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

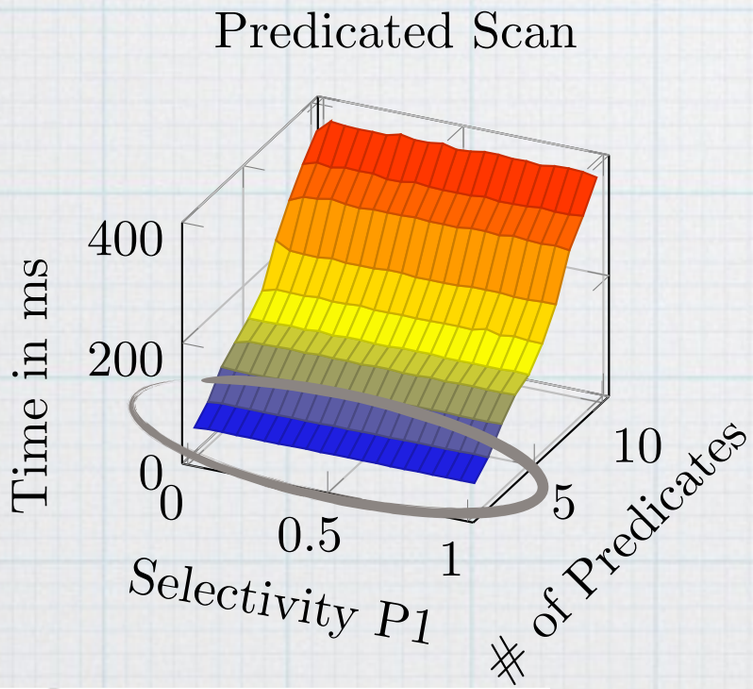
D B S E

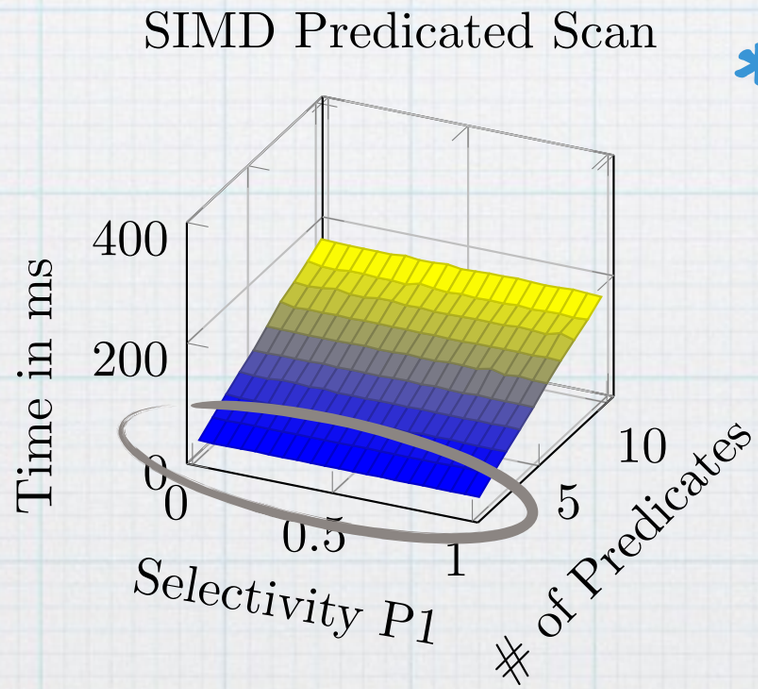
Number of Predicates







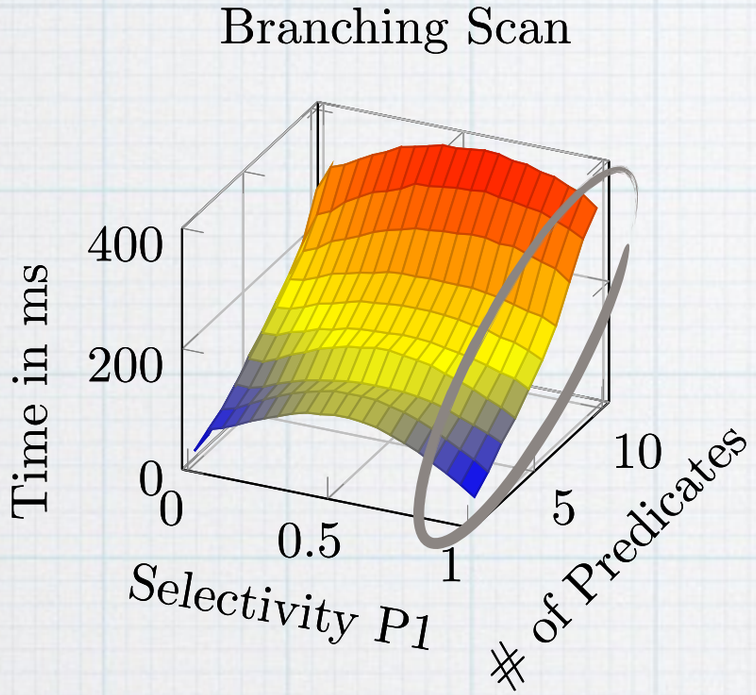


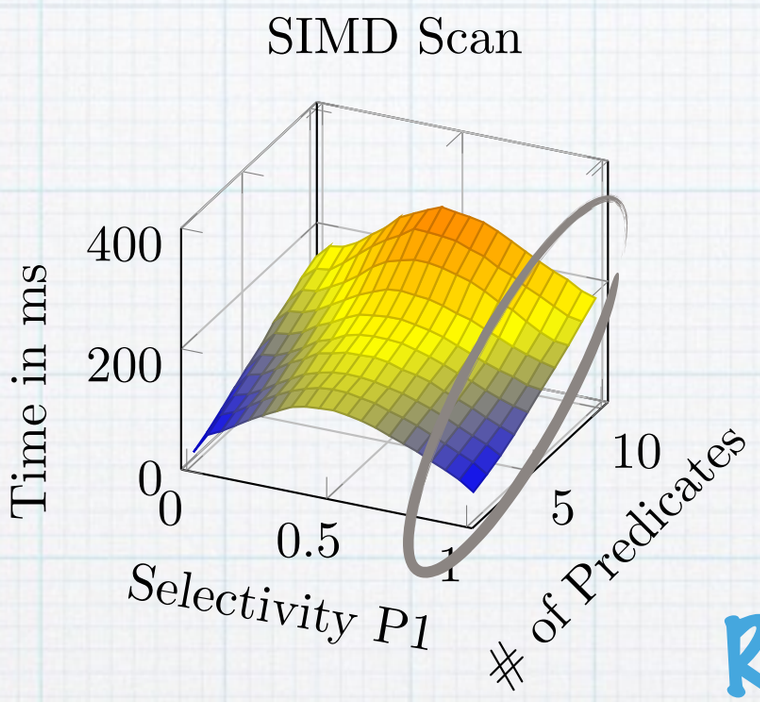


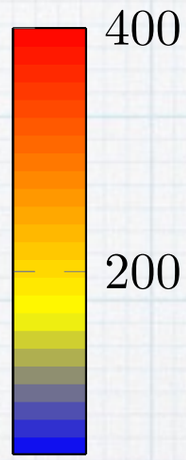
Results:

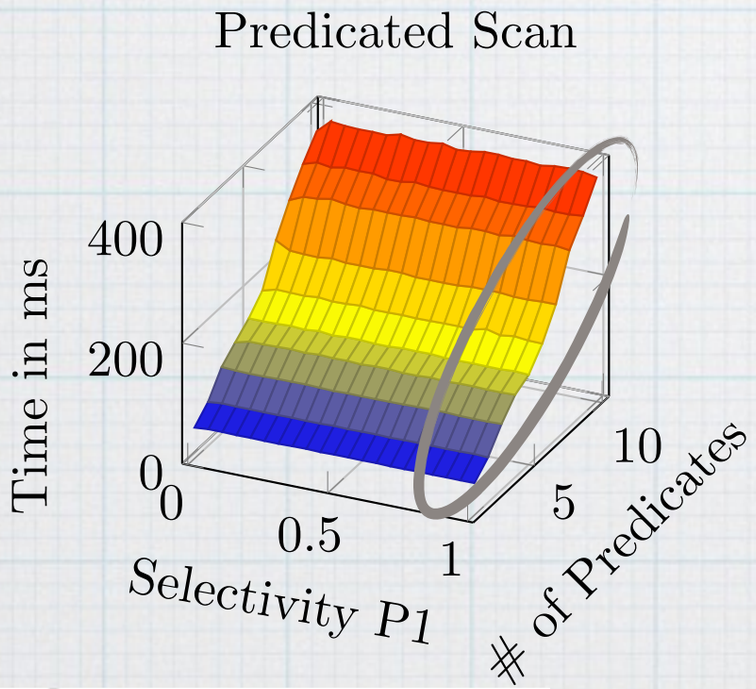
* For one predicate SIMD does not pay out

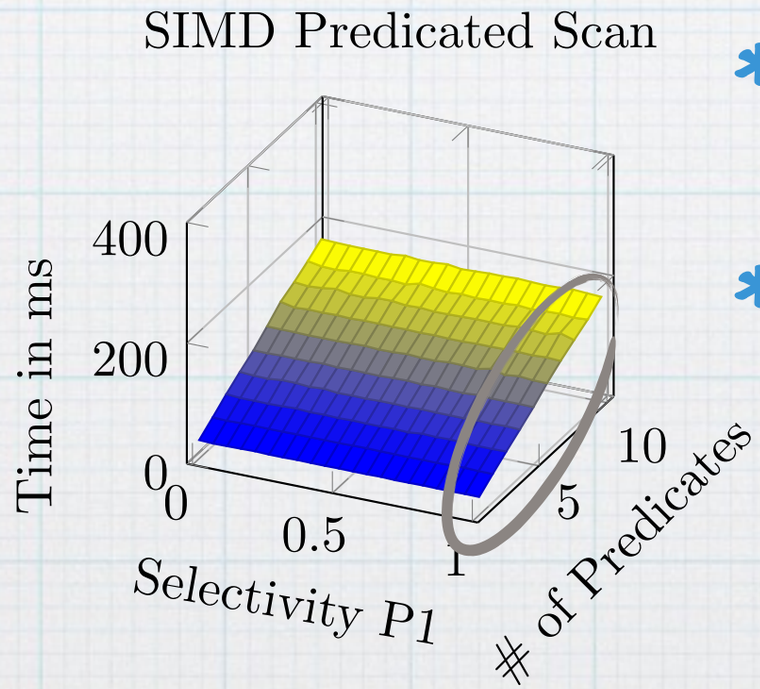
Number of Predicates











Results:

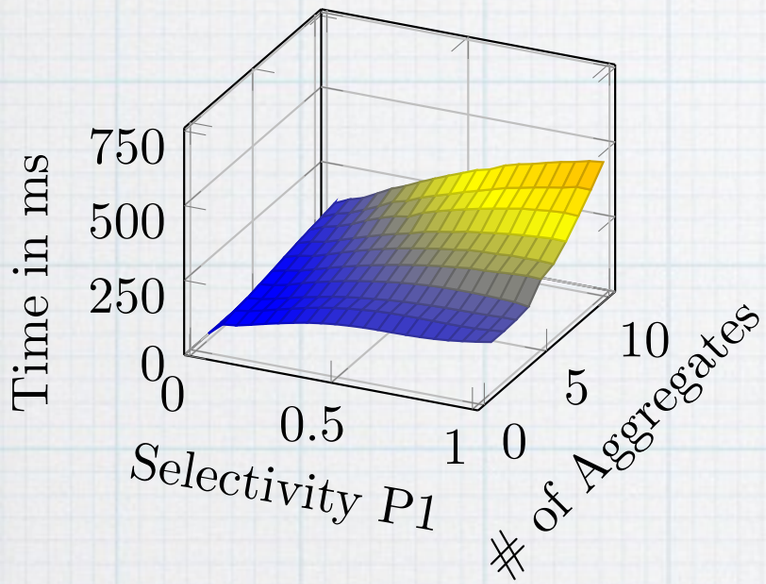
- For one predicate SIMD does not pay out
- The more predicates, the better SIMD

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

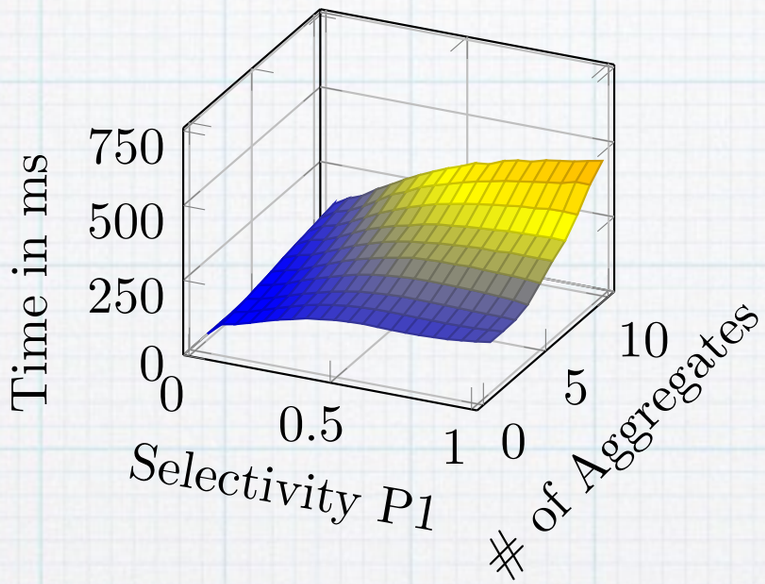
DBSE

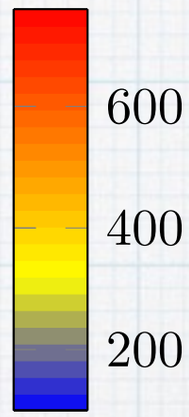
Work Inside the Loop

Branching Scan

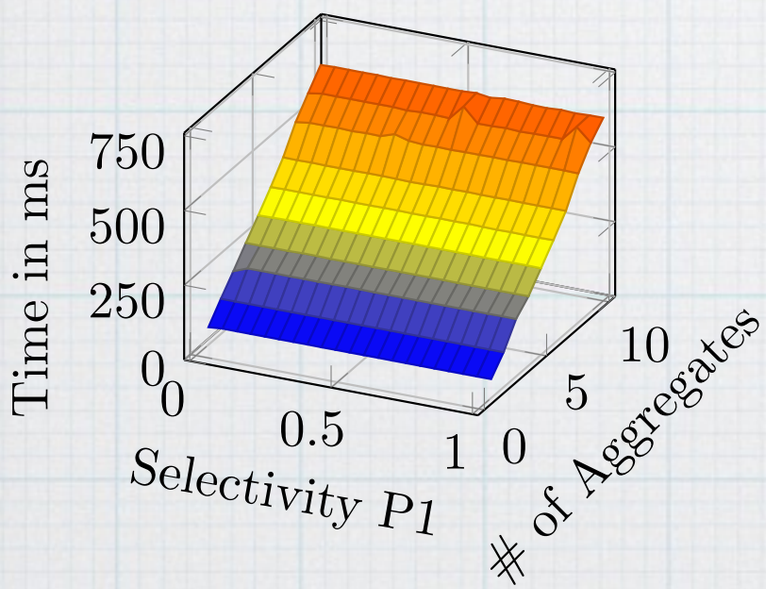


SIMD Scan

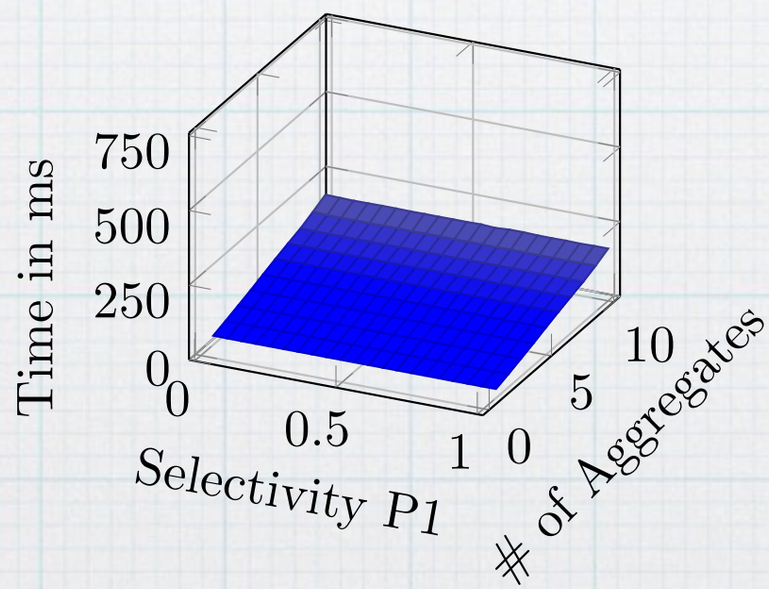




Predicated Scan



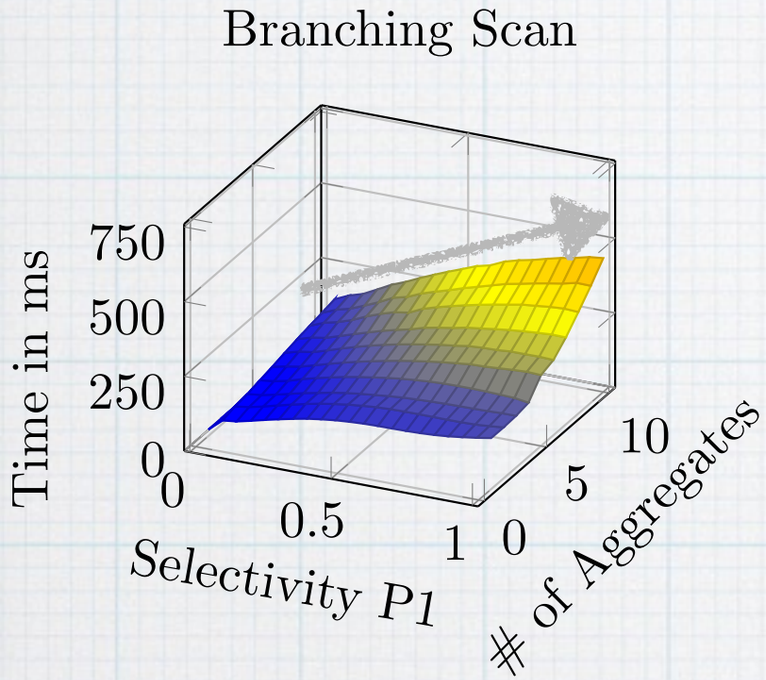
SIMD Predicated Scan

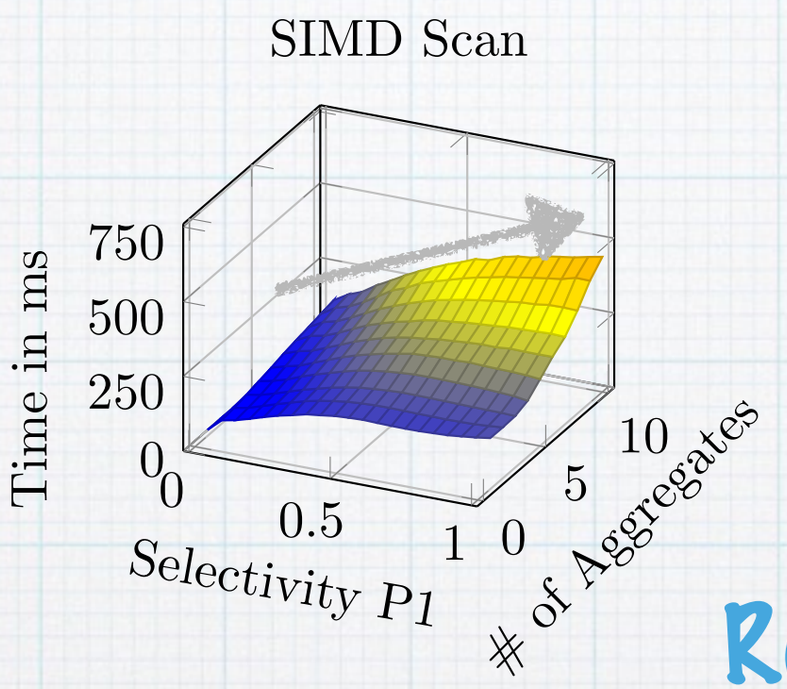


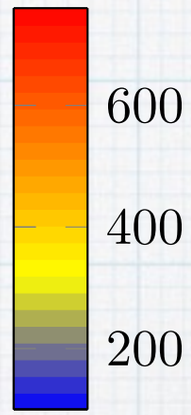
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

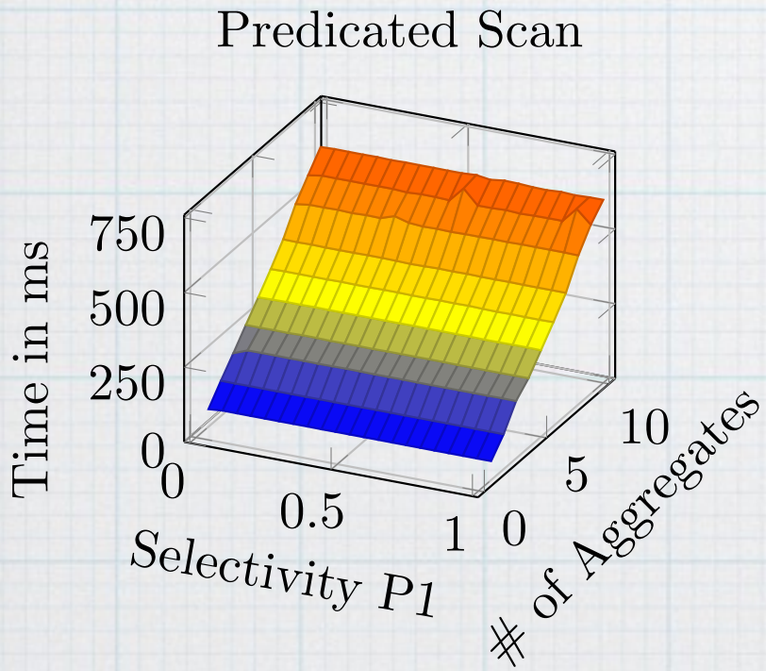
D B S E

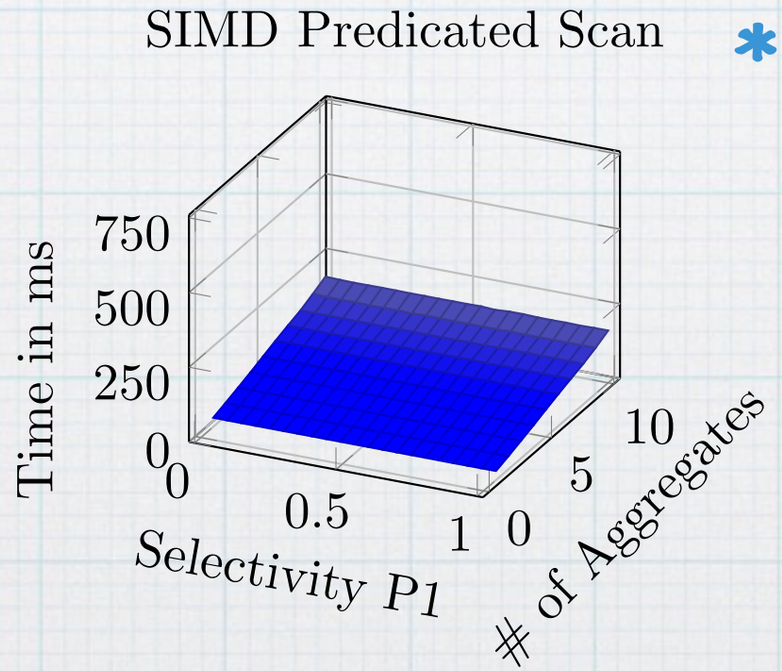
Work Inside the Loop







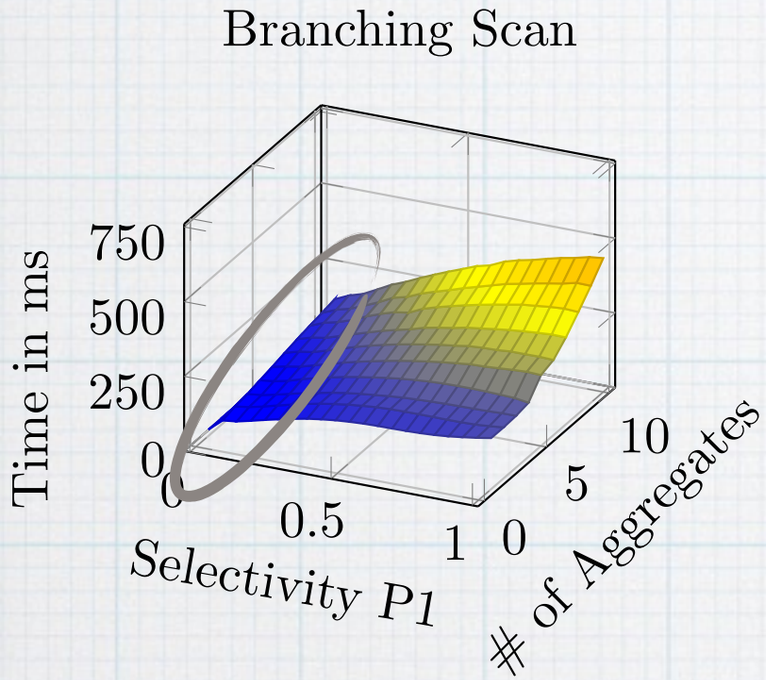


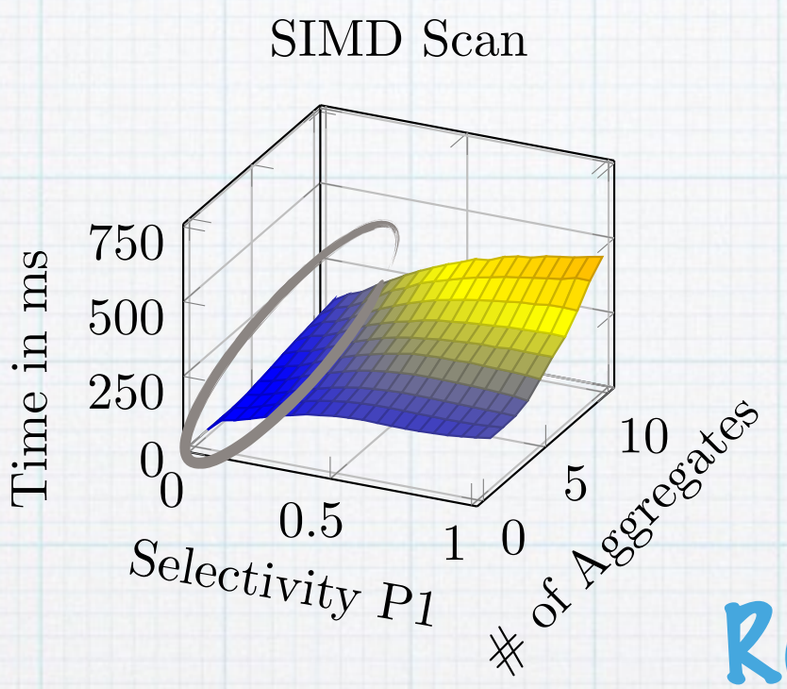


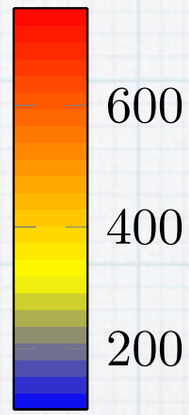
Results:

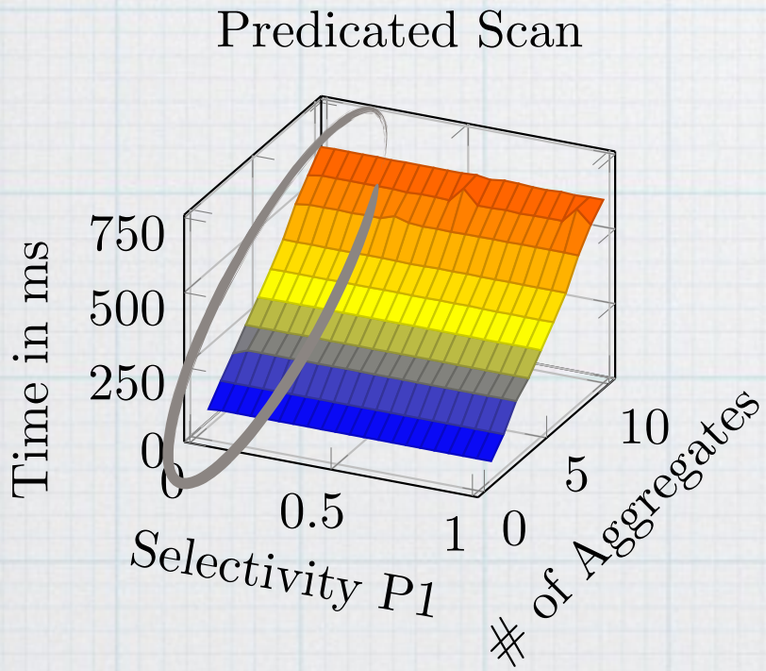
* More aggregates, less impact of branch misprediction

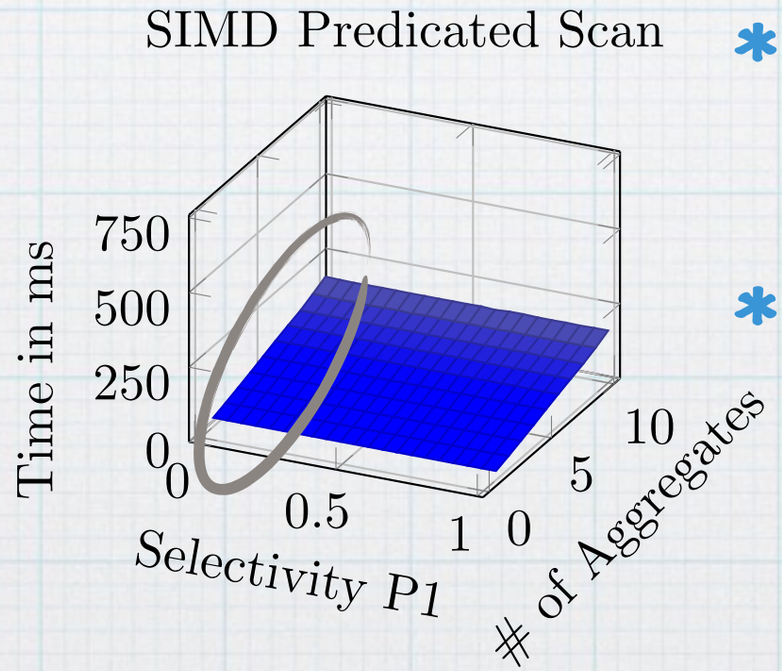
Work Inside the Loop









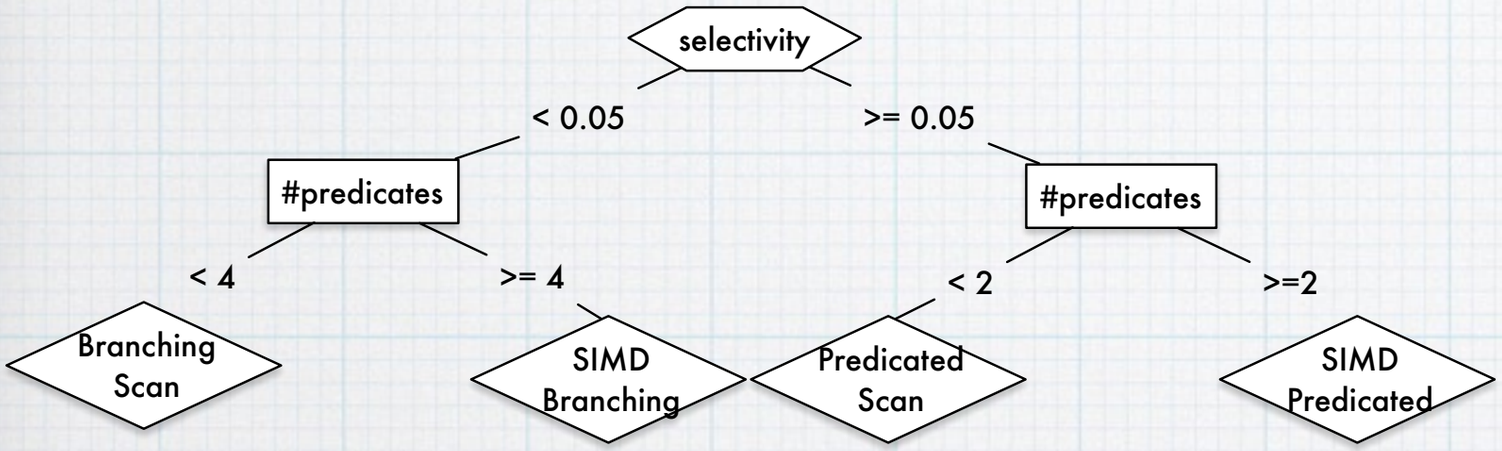


Results:

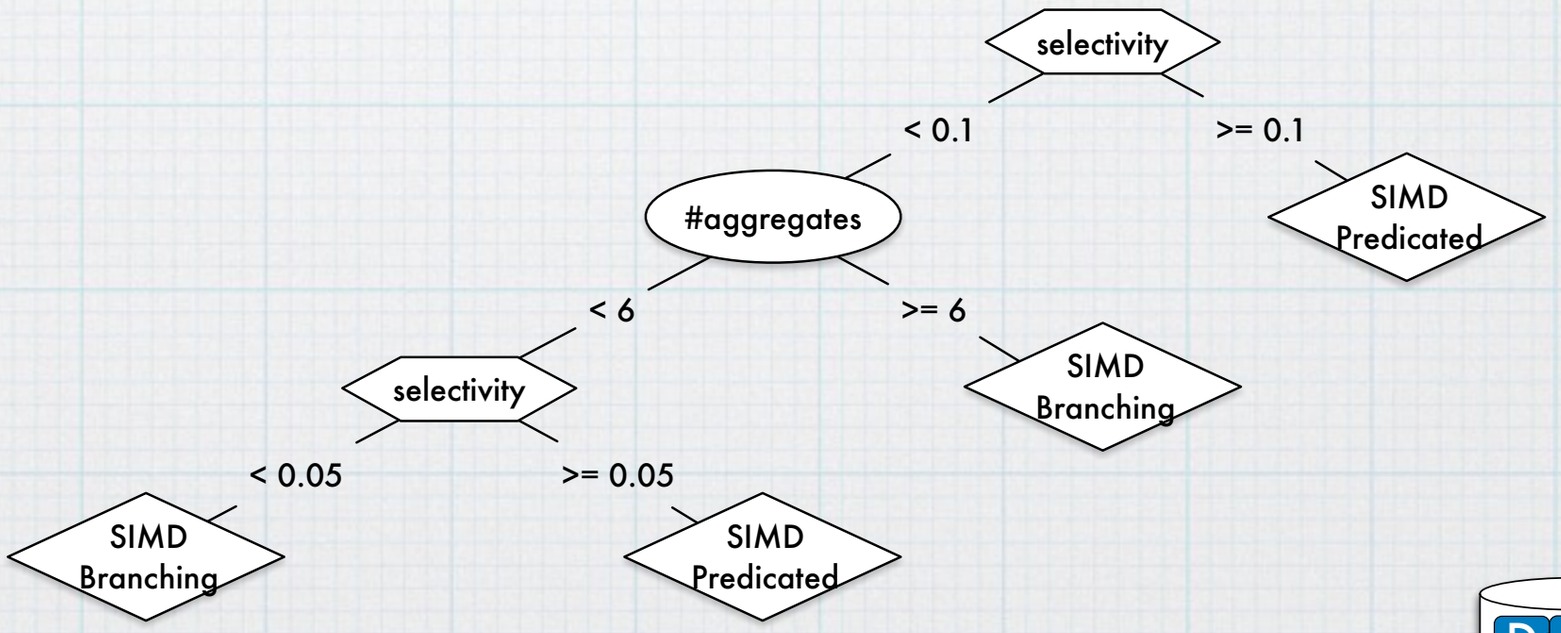
* More aggregates, less impact of branch misprediction
* The more aggregates, the better branching scans for low selectivity

Decision Trees

Number of Predicates



Number of Aggregates



Conclusion

* Increasing number of aggregates slows down predicated variants
* SIMD outperforms scalar variants for several predicates
* Pipeline code for filter-&-aggregate pipelines[1]
* Decision trees as a result of our evaluation in the paper

Future Work

* Hash table put / probe (joins, groupings)
* Automatic calibration for query compilation

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF

[1] http://git.iti.cs.ovgu.de/dbronesk/BTW-Pipeline-Variants

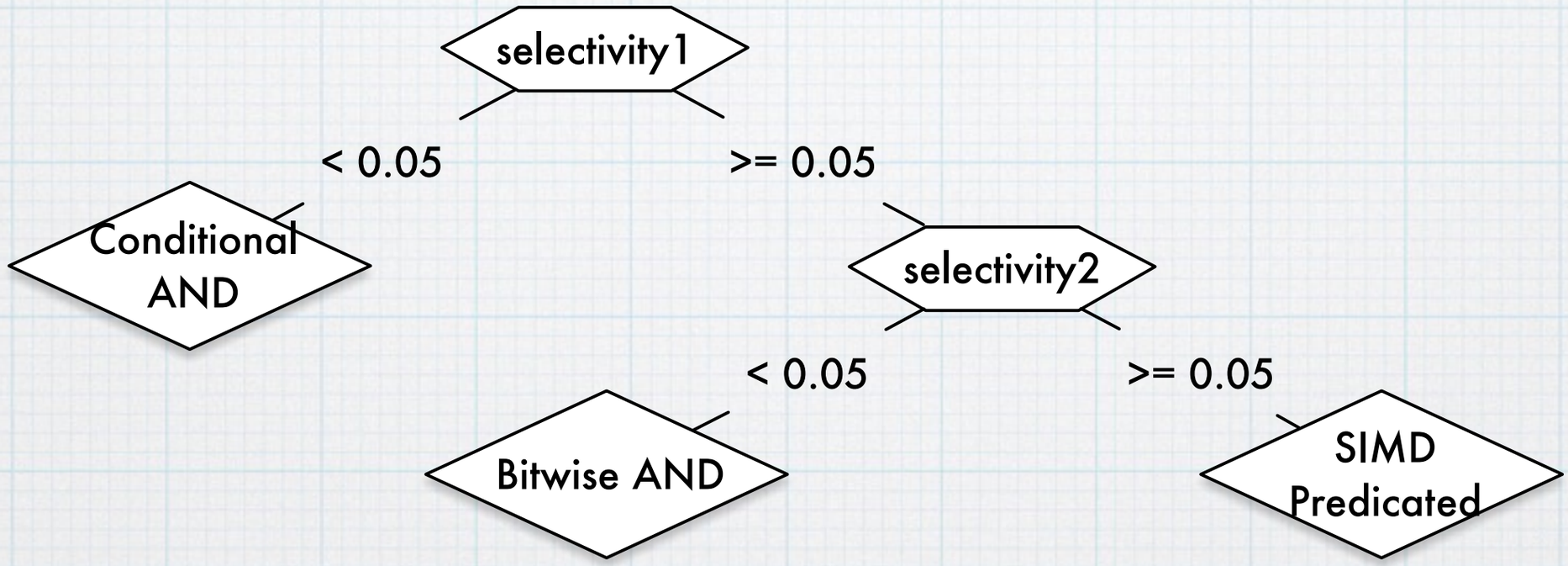
DBSE

References

[BBS14] David Broneske, Sebastian Breß, and Gunter Saake. Database Scan Variants on Modern CPUs: A Performance Study. In Proceedings of the 2nd International Workshop on In-Memory Data Management and Analytics (IMDM), Lecture Notes in Computer Science, pages 97-111. Springer, 2014

[ZR02] Jingren Zhou, Kenneth A. Ross: Implementing database operations using SIMD instructions. In: SIGMOD. Pp. 145-156, 2002.

Selectivity of Two Predicates



Selectivity of Two Predicates