

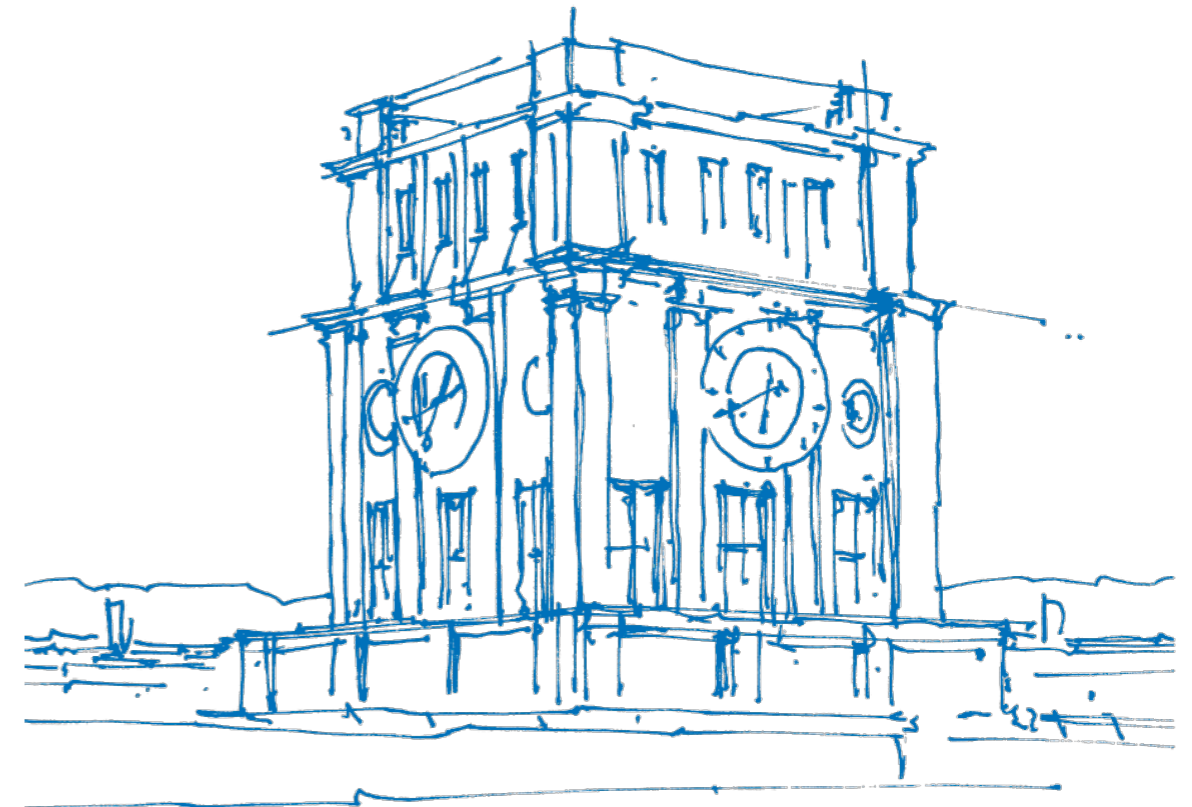
Bonusproject 2, Execution Engines

Timo Kersten

Technische Universität München

Faculty for Computer Science

Chair for Database Systems

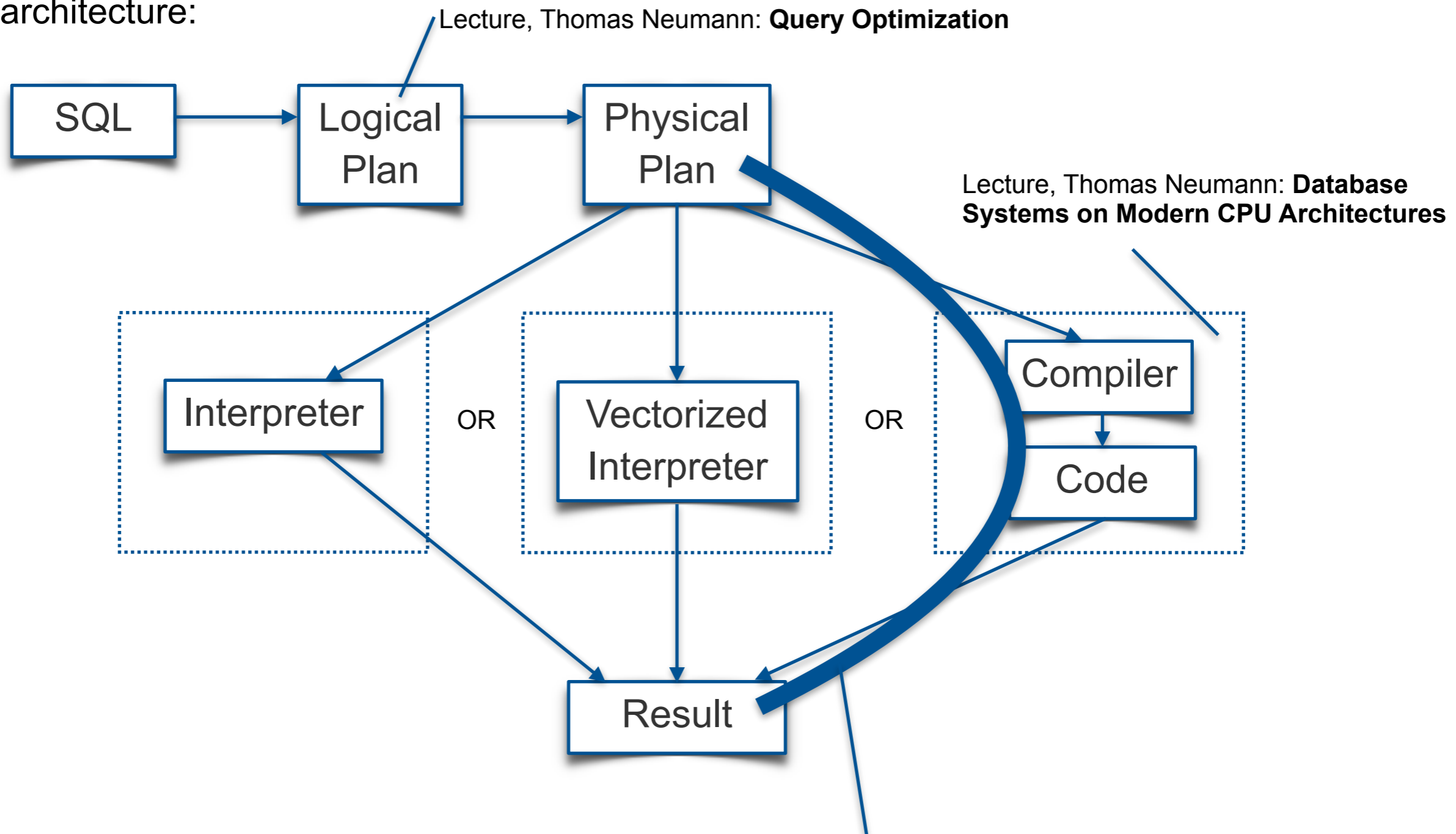


TUM Uhrenturm

DBMS Execution Engines

Execution Engines

DBMS architecture:



Lab Course, A. Kohn und T. Neumann: Database Implementation

Traditional Interpreters (in DBMS)

Volcano style interpreters

G. Graefe, Volcano - An Extensible and Parallel Query Evaluation System

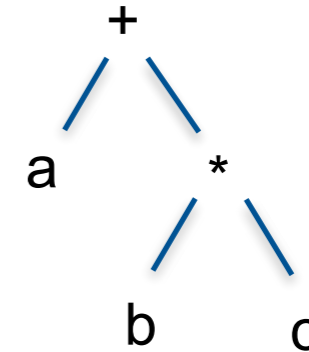
Example: Expression evaluator: $a + b * c$

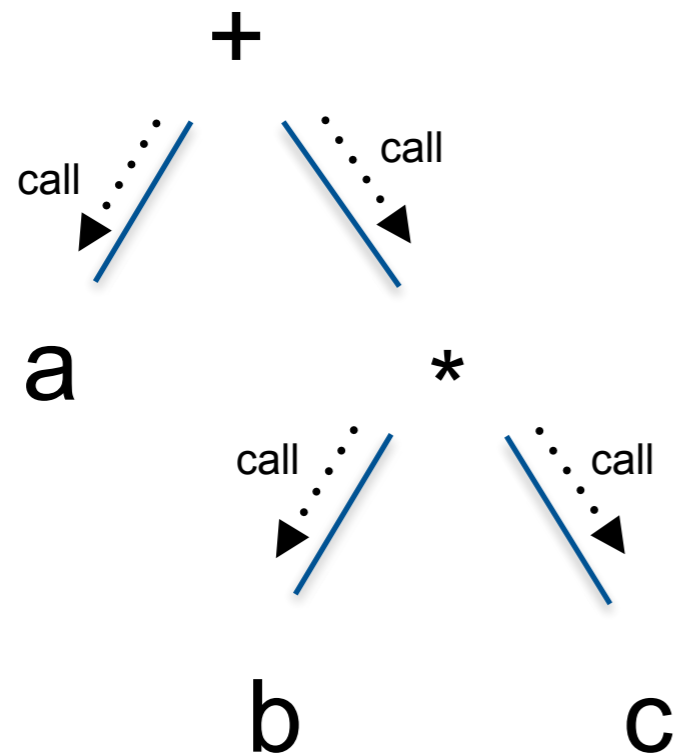
```
class Operator;
class BinaryOperator;
class Plus;
class Mul;

class Operator {
public:
    virtual int compute();
};

class BinaryOperator : public Operator {
protected:
    Operator *left, *right;
};

class Plus : public BinaryOperator {
    int compute() override {
        auto l = left->compute();
        auto r = right->compute();
        return l + r;
    }
};
```





Many virtual function calls

- ➔ Register value saving, memory traffic
- ➔ Hard to predict branches, miss penalty 15c
- ➔ Extra instructions

Vectorized Interpreter

Idea: Penalties are on a per call basis. Let's pass multiple elements per call
-> Amortize call cost over batch

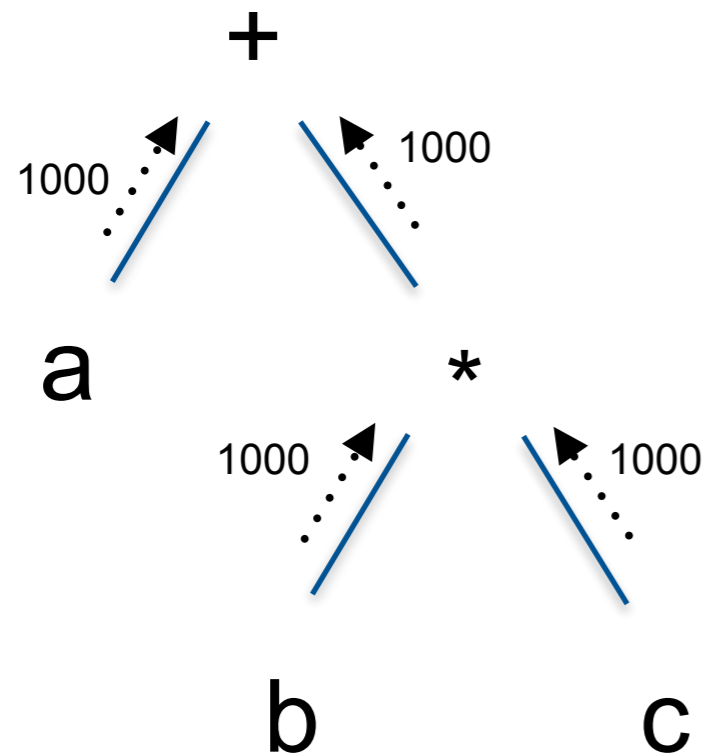
```
#include <vector>

class Operator;
class BinaryOperator;
class Plus;

class Operator {
public:
    virtual vector<int> compute();
};
...
class Plus : public BinaryOperator {
    vector<int> compute() override {
        auto l = left->compute();
        auto r = right->compute();
        vector<int> result;
        for (int i = 0; i < l.size(); ++i)
            result.push_back(l[i] + r[i]);
        return result;
    }
};
```

← One call per batch

← Tight loop over elements



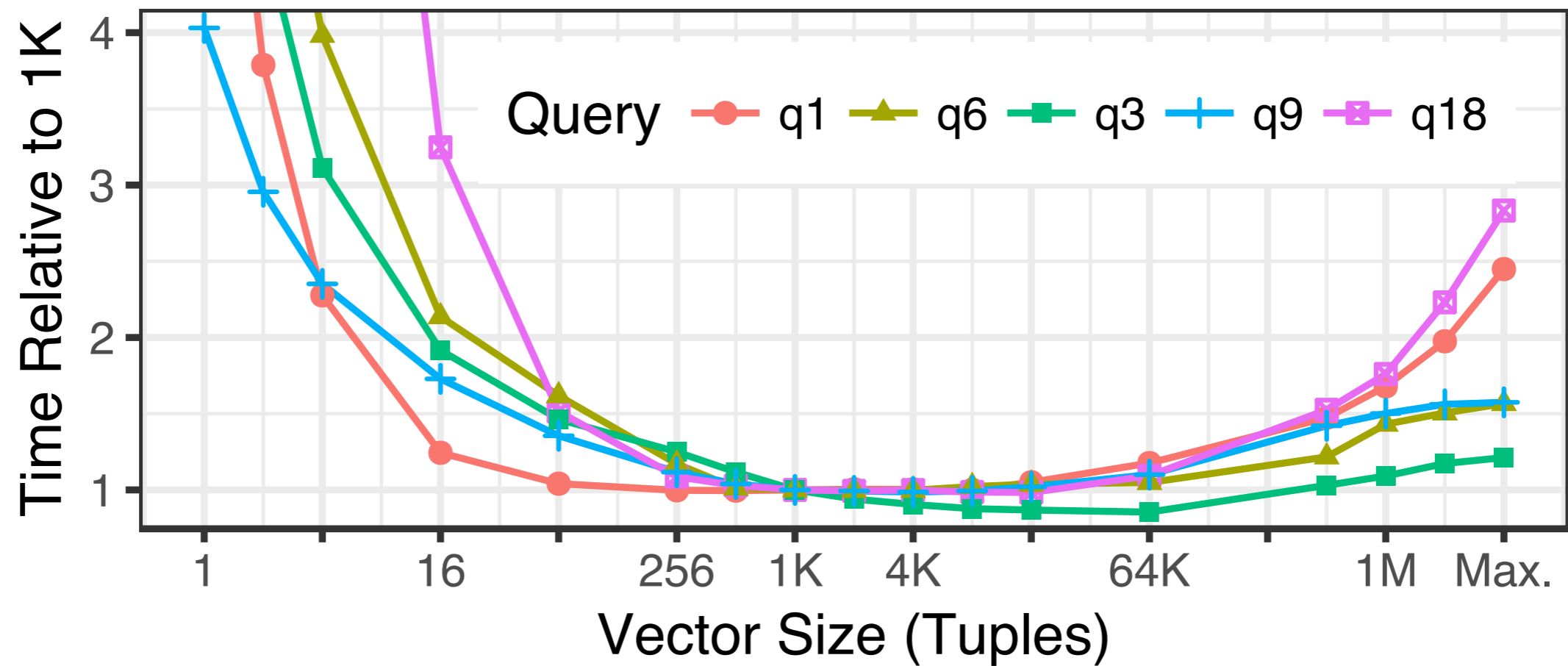
Amortizes call overhead.

Complications:

- Memory traffic
- Control flow, e.g. selection, join, grouping
 - ➔ Selection vectors
 - ➔ Sparsely populated vectors after operation, less effective amortisation
- Type combinations, e.g. combined hash table keys

Optimal Vector Size

Runtimes of selected TPC-H queries
Sf=1



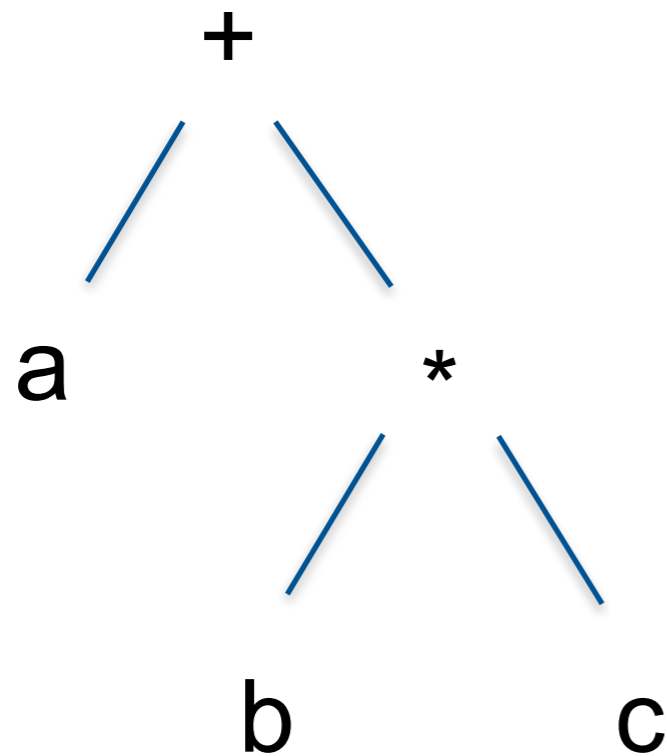
Trade-off between amortisation and cache capacity

Vectorized Interpreter: Pros

Remember: Some restrictions on type combinations

But some advantages:

- Call overhead is amortised
- Each primitive can use the full power of C to work on multiple elements
 - That means we can easily use hardware features, e.g. SIMD, hashing instructions, prefetching.
- Everything precompiled: Query execution can start right away.



```
int compiledFun(int a, int b, int c) {  
    return a + b * c;  
}
```

Pro:

- No virtual function calls
- Intermediate values can be kept in registers
- Compiler can choose minimal number of instructions
- Data flow becomes control flow

Cons:

- Compile time
- How to use SIMD etc.?

If one wants to build a new query engine today,
which paradigm should one use?

Vectorwise vs. HyPer?



MonetDB/X100: Hyper-Pipelining Query Execution, Boncz et al.,
CIDR 2005

Other systems with vectorized architecture:
Quickstep, Snowflake, ...



Efficiently Compiling Efficient Query Plans for Modern Hardware, Neumann, VLDB 2011

Other systems with query compilation:
Peloton, Microsoft Hekaton, Spark, ...

Not directly comparable:

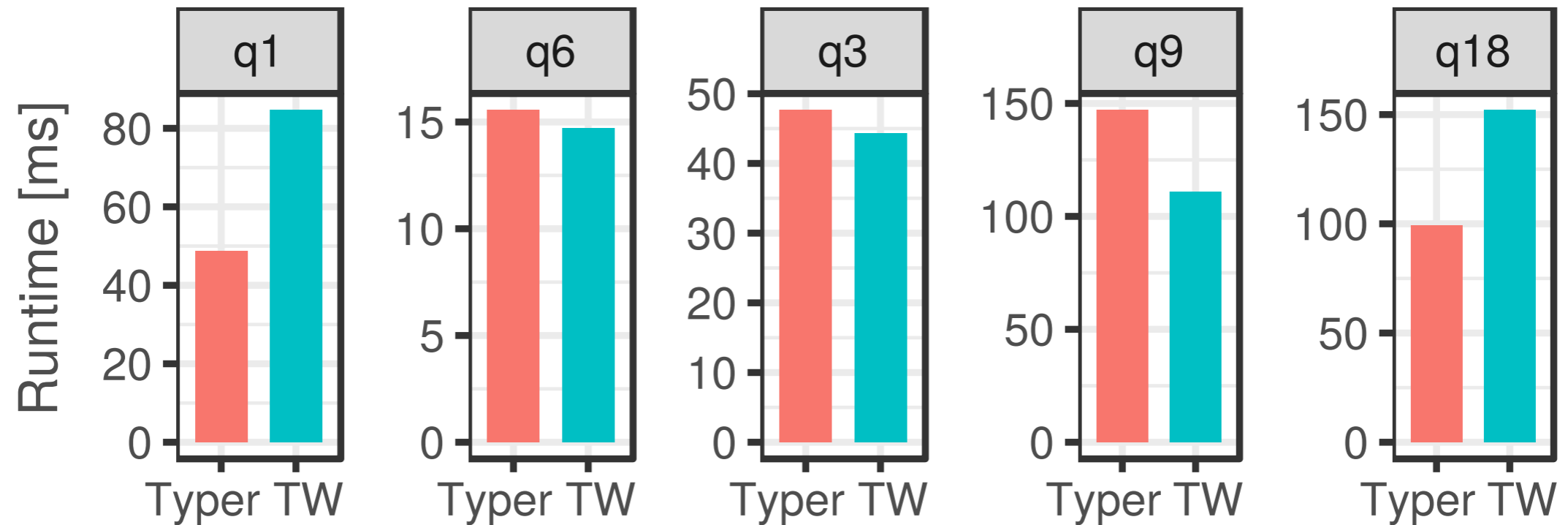
- Different storage/compression schemes
- Different query processing algorithms and data structure
- Different parallelisation frameworks
- Different query optimisers
- ...

Tectorwise vs. Typer

Back to back comparison: Implementation of both paradigms

- Same storage/compression schemes (mmap-ed uncompressed columns)
- Same query processing algorithms and data structure (down to identical hash tables)
- Same parallelisation frameworks (morsel-driven parallelization)
- Same query optimisers (identical query plans)
- ...

TPC-H (SF=1, 1 thread, no SIMD)



Main cost from:

q1: Expression Evaluation

q6: Selection

q3: Join with small hts

q9: Join with big hts

q18: Grouping

Microarchitectural Analysis: TPC-H q1

```
select  l_returnflag, l_linestatus,  
        sum(l_quantity),  
        sum(l_extendedprice),  
        sum(l_extendedprice * (1 - l_discount)),  
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)),  
        avg(l_quantity),  
        avg(l_extendedprice),  
        avg(l_discount),  
        count(*)  
from    lineitem  
where   l_shipdate <= date '1998-12-01' - interval '90' day  
group by l_returnflag, l_linestatus  
order by l_returnflag, l_linestatus
```

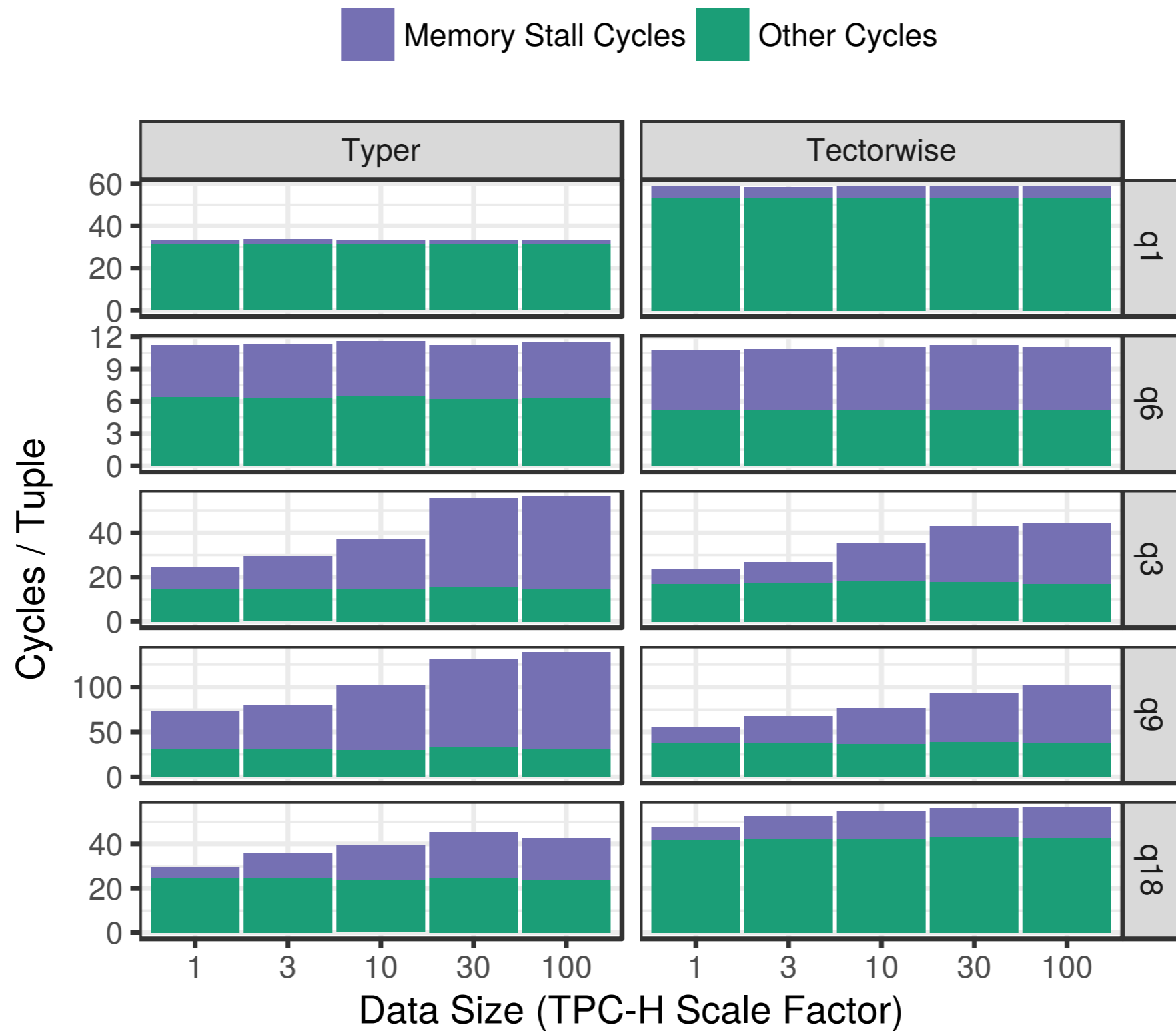
	cycles	IPC	instr.	L1 miss	LLC miss	branch miss	mem. stall cycles
Typical	34	2.0	68	0.6	0.57	0.01	1.8
Blockwise	59	2.8	162	2.0	0.57	0.03	5.2

Microarchitectural Analysis: TPC-H q9

```
select ...  
from lineitem, orders, partsupp,  
      supplier, part, nation  
where s_suppkey = l_suppkey  
and ps_suppkey = l_suppkey  
and ps_partkey = l_partkey  
and p_partkey = l_partkey  
and o_orderkey = l_orderkey  
and s_nationkey = n_nationkey  
and p_name like '%green%'
```

	cycles	IPC	instr.	L1 miss	LLC miss	branch miss	mem. stall cycles
Typical	74	0.6	42	1.7	0.46	0.34	42
Blockwise	56	1.3	76	2.1	0.47	0.39	18

Stall Cycles



- < *Computation*: compiled code can keep data in CPU registers
 - > *Parallel data access*: vectorisation is better at generating parallel cache misses
 - = *SIMD*: easier to apply, though gains are limited
 - = *Parallelization*: both scale well
 - = *Hardware platforms*: Skylake, Knights Landing, and AMD Ryzen are similar
- ➔ overall performance similar on OLAP, no clear winner

Other Aspects Are Probably More Important



- < *OLTP*: compilation results in fast stored procedures
- < *Language support*: compilation enables seamless integration of different languages
- > *Compile time*: vectorisation has no JIT-compilation overhead
- > *Profiling*: Runtime can easily be attributed to vectorised primitives
- > *(Micro-)adaptivity*: vectorisation allows primitives to be swapped mid-flight

for optimal performance, one needs a hybrid engine that combines both approaches

If You Are New to TUM



www.in.tum.de/academic-advising

Have a look at the “Let’s talk about ...” series.



Cover Page > Current Students > Advising and Support > Academic Advising

Academic Advising

Student advisors at TUM Informatics are available for all students and covers topics including (but not limited to):

Adademic Student Advising:

- Transferring credits to and between TUM Informatics programs
- Examination regulations
- Individual study plans due to leave of absence or Master’s Informatics part-time program
- Support for disabled or chronically ill students
- Studies, internships, and thesis research abroad
- Repeating exams and catching up after falling behind the regular program schedule
- Dealing with lack of motivation
- Learning methods

International Student Advising:

Advising on special situations and issues that are relevant to international students is also available, including the following topics:

- Problems with resident permits/visas
- Scholarships and financial problems
- Guidance on working while studying in Germany
- Information about the medical care System in Germany
- First contact for international students in difficulties
- Homesickness and adjusting to life in Germany
- Advice for getting to know more German students or other students

Admissions:

- Formal requirements for Master’s program admissions (e.g. which Bachelor degrees are accepted)
- All questions related to the admissions process

Don’t forget to check out our [Tips for Successful Studies!](#)

Overview important dates Winter Semester

Winter Semester	Event	Deadline Winter Semester 2018/2019
October	Orientation program for new international degree students https://www.tum.de/nc/en/studies/international-students/refugees/	www.in.tum.de/en/orientation-program-masters/
	Language Café	regularly every Wednesday, 3-5.30 pm from 17.10.18
	Open House Day 2018	13th October 2018
	Collective opening hour	15th October 2018, 10 am, MI 00.13.009A
November	Let’s talk about - Courses & Exams @in.tum	29th November 2018, 12 - 2 pm, Interimshörsaal 2
December	Let’s talk about - IDP @in.tum	13th December 2018, 12 - 2 pm, Interimshörsaal 2
January	Information event Master’s Thesis	31st January 2019, 12 - 2 pm, Interimshörsaal 2

Department of Informatics
Boltzmannstr. 3
85748 Garching

Learn more
Please always write with your real name from your TUM university account!
Just drop in during our opening hours **without previous notice!**
During the **first week of lectures** usually many students come. If your concern still has time come later, otherwise come early and bring some time with you.
During our opening hours we are usually unavailable via telephone.
[Tips for Successful Studies](#)

Vivija Simić

Head of Academic Advising
office: 01.08.039
email: study-advising@in.tum.de
phone: +49 89-289-17296
[consultation hours](#)

Sibylle Roden-Kinghorst

Academic Student Advising, Departmental contact for chronically ill and disabled students
office: 02.07.039
email: study-advising@in.tum.de
phone: +49 89-289-17284
[consultation hours](#)

Lena Krone

Z-order curve: https://aws.amazon.com/blogs/database/z-order-indexing-for-multifaceted-queries-in-amazon-dynamodb-part-1/?sc_channel=sm&sc_campaign=zackblog&sc_country=global&sc_geo=global&sc_category=rds&sc_outcome=aware&adbsc=awsddbblog_social_20170517_72417147&adbid=864895517733470208&adbpl=tw&adbpr=66780587

Bloom Filter <https://www.kdnuggets.com/2016/08/gentle-introduction-bloom-filter.html>

Execution Engine Comparison

Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask, Kersten et al., PVLDB 2018