

Lecture Series on
Database Research

Winter Semester 2023 / 2024
Introduction & Hardware Efficient Stream Processing

Prof. Dr. Tilmann Rabl, Prof. Dr. Felix Naumann
Data Engineering Systems
Hasso-Plattner-Institut

Attention!

This lecture is recorded and will be available in Tele-Task.

We do not record the videos in online sessions, feel free to enable video sharing.

You are not allowed to record the lecture or take screenshots or pictures.



This Lecture

1. Motivation
 - DB Research
 - This lecture
 - Speakers
2. Course Organization
 - Exercises
 - Grading
 - Registration
3. Stream Processing
 - Basics & Code Generation
 - Portable SIMD Code

Who am I?



- At Hasso Plattner Institute
- Department of U Potsdam
- In the vicinity of Berlin

- Tilmann Rabl
- Chair for Data Engineering Systems



Why Database Research – my personal view

Why I chose to do DBS research:

- Database systems are examples of large complex software systems
 - Like OS, compilers, computer games
 - Building a DBS touches all topics of computer science
- Database systems are at the core of all major businesses
 - Safe and efficient processing is relevant everywhere
 - Significance of data processing is only increasing



Why I got stuck in DBS research

- Fun research and ongoing challenges
- Great community

Why this course

- Germany has an active and productive DB research community
- Best DB conferences always with good German participation
 - E.g., VLDB, SIGMOD
- However, there is little awareness and exchange on student level
- We want to make you aware of excellent German database research
 - Foster exchange
 - Show opportunities

Presentations by *distinguished researchers* from the *German database research* community (and friends and family).

- Some universities also offer the course locally



Presenters I



24.10.

Pinar Tözün

ITU Copenhagen

Hardware Parallelism &
Transaction Processing Systems



7.11.

Matthias Böhm

TU Berlin

System Infrastructure for
Data-centric ML Pipelines

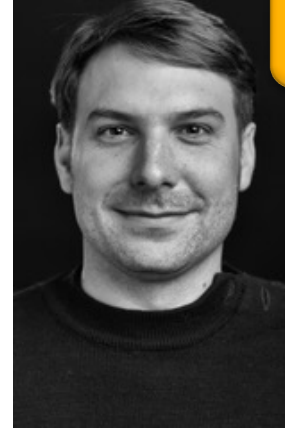
Presenters II



14.11.

Torsten Grust
Uni Tübingen

A Fix for the Fixation on Fixpoints
(Rethinking Iteration in SQL)



21.11.

Hannes Mühleisen
CWI & Uni Nijmegen

In-Process OLAP

Presenters III



28.11.

Stefanie Scherzinger

Uni Passau

Challenges with JSON Schema Data Modeling



05.12.

Viktor Leis

TUM

Commoditizing Data Analytics in the Cloud

Presenters IV



12.12.

Wolfgang Lehner

TU Dresden

Flexible Vector Processing for
Data Science Engines



19.12.

Zsolt István

TU Darmstadt

Software-Defined Data Protection:
Low Overhead Policy Compliance at
the Storage Layer is Within Reach!

Presenters V



09.01.

Matthias Weidlich

HU Berlin

Pushing Computation to the
Sources: On Distribution in
Complex Event Processing



25.01.

Ziawasch Abedjan

Leibniz Uni Hannover

Data Cleaning

Presenters VI



23.01.

Felix Naumann

HPI

Data Profiling



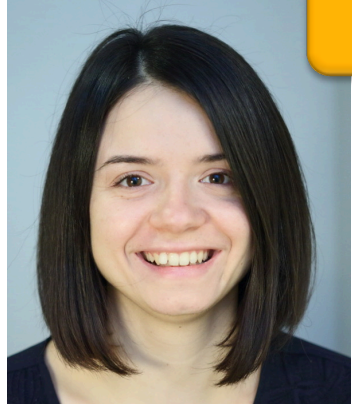
30.01.

Carsten Binnig

TU Darmstadt

Towards Learned
Database Systems

Presenters VII

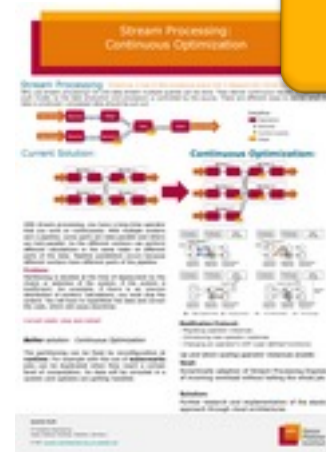


06.02.

Jana Giceva

TUM

TBD



09.02.

Poster Session

HPI

Event Space L

Course Logistics at HPI

Logistics

Lecture

- Tuesdays, 17-18:30, L.E-03
- Bachelor & Master, 3 ECTS

Contact (at HPI)

Prof. Dr. Tilmann Rabl, Prof. Dr. Felix Naumann

- e-mail: tilmann.rabl@hpi.de, felix.naumann@hpi.de

Deliverables I

Lecture Summary

- Summarize one presentation
- In group or individually
- Contents
 - Introduction of speaker
 - Engaging overview of talk
 - Goal, problem, solution
 - Background
 - Main ideas, methodology, approach
 - Main results
 - Summary
- Will be added to the course website (should be 10-15 min reading time)
 - Cf. Morning Paper - <https://blog.acolyer.org/>
- We will ask the presenter to review the summary
- Plain HTML formatting
- Deadline 2 weeks after presentation
- 50% of the final grade

Deliverables II

- Poster project
 - A1 poster
 - Plenary presentation at Feb 9

- Bachelor Students
 - Based on one of the lectures
 - Related work overview, lecture summary,
 - As detailed as possible

- Master Students
 - Research project proposal
 - Extend the technology or methodology of one more lectures
 - Goal, problem, solution & connection to the lecture

Registration

- Organization through HPI Moodle
 - <https://moodle.hpi.de/course/view.php?id=630>
 - Sign up now!

- Groups
 - Will be assigned randomly
 - TBD...

Grading in a Nutshell

- Lecture Summary
 - Team project
 - 50% of total points
- Poster
 - Individual project
 - 50% of total points
- Graded by the teachers

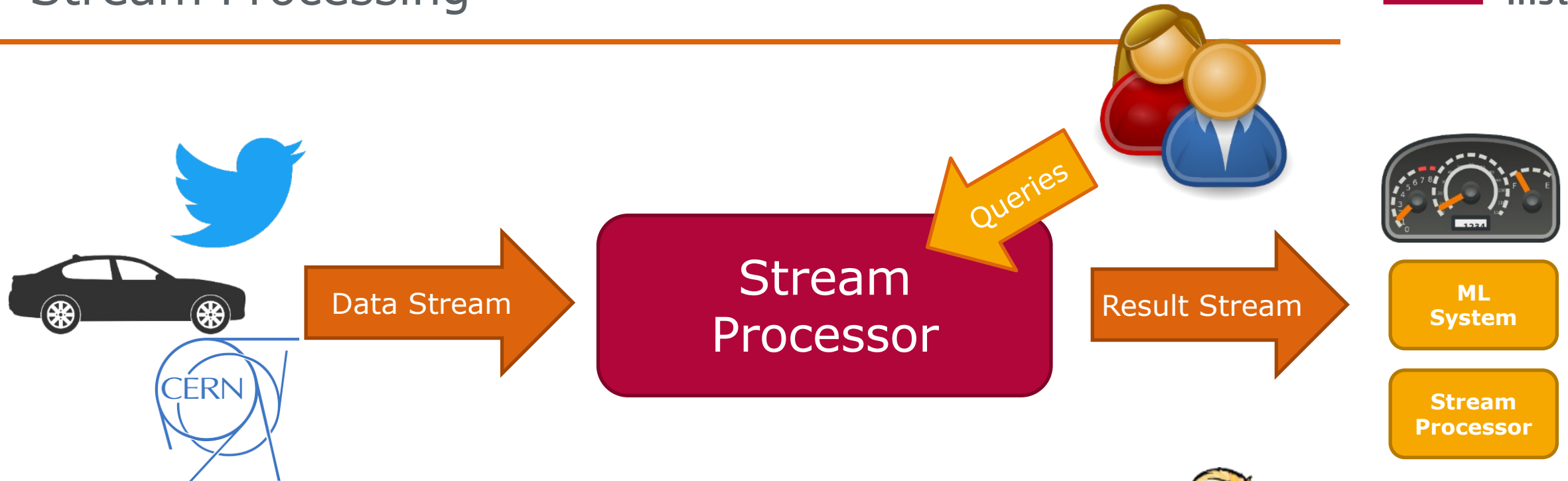
Code of Conduct

- Asking questions is greatly encouraged
 - Discuss questions with each other (except exams)
- The limits of collaboration
 - Plagiarism, copying, or other forms of dishonesty **will result in failing the course**
- Communication
 - Write professional and polite emails
 - Use netiquette in forum, email, chats, etc.
- Generally
 - Treat everyone with respect and consideration -> especially important for online settings
 - This course and the university in general should be safe spaces for everyone

Stream Processing

End-to-end Hardware-conscious Data Processing

Stream Processing



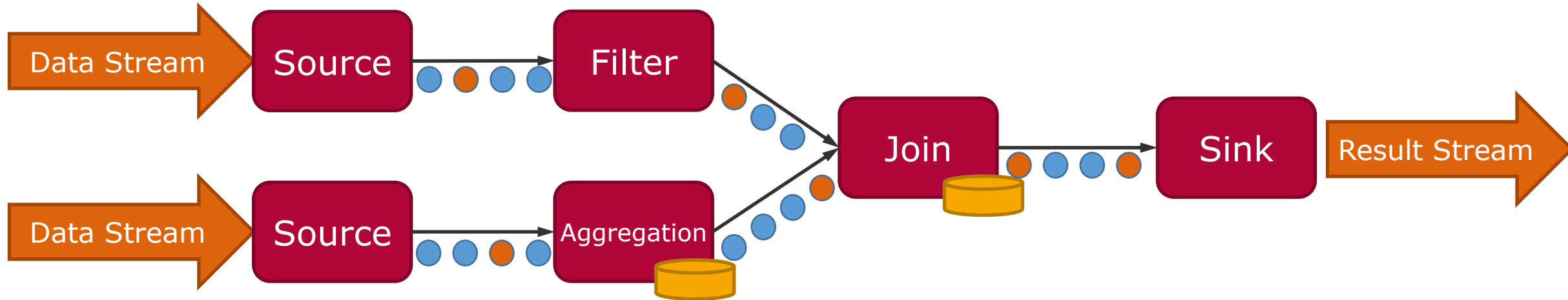
Challenge

- Potentially unlimited data set
- Many different queries
- Low latency, continuous results





Typical Systems



Stream Processor Research



Dataflow

-  Operators
-  Records
-  Control events
-  State

Research topic examples

- Application (e.g., incremental instead of batch)
- Operator-level (e.g., aggregation, join)
- Semantics (e.g., window types, watermarks)
- Execution strategy (e.g., multi-query processing, compilation)
- Hardware optimizations (e.g, PMem, RDMA)

Why Hardware Research?

not possible!

scale-up system

Four Billion Records per Second! What is Behind Alibaba Double 11 - Flink Stream-Batch Unification Practice during Double 11 for the Very First Time

Alibaba Clouder December 2, 2020 1,889 0

Grizzly: Efficient Stream Processing Through Adaptive Query Compilation

Philipp M. Grulich¹ Sebastian Breß¹ Steffen Zeuch^{1,2} Jonas Traub¹
 Janis von Bleichert¹ Zongxiong Chen² Tilmann Rabl³ Volker Markl^{1,2}
¹Technische Universität Berlin ²DFKI GmbH ³HPI, University of Potsdam

ABSTRACT

Stream Processing Engines (SPEs) execute long-running queries on unbounded data streams. They follow an interpretation-based processing model and do not perform runtime optimizations. This limits the utilization of modern hardware and neglects changing data characteristics at runtime.

In this paper, we present *Grizzly*, a novel adaptive query compilation-based SPE, to enable highly efficient query execution. We extend query compilation and task-based par-

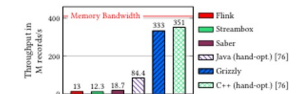


Figure 1: Yahoo! Streaming Benchmark (8 Threads).

such as Flink [15] and Storm [66] scale-out executions to achieve high throughput and low-latency. However, recent



scale-out system



16-core instance @ \$1/h

~42k events/s per server

93750 VMs @ \$2.25 million/day

1.5 million CPUs

> 100 million events/s per server

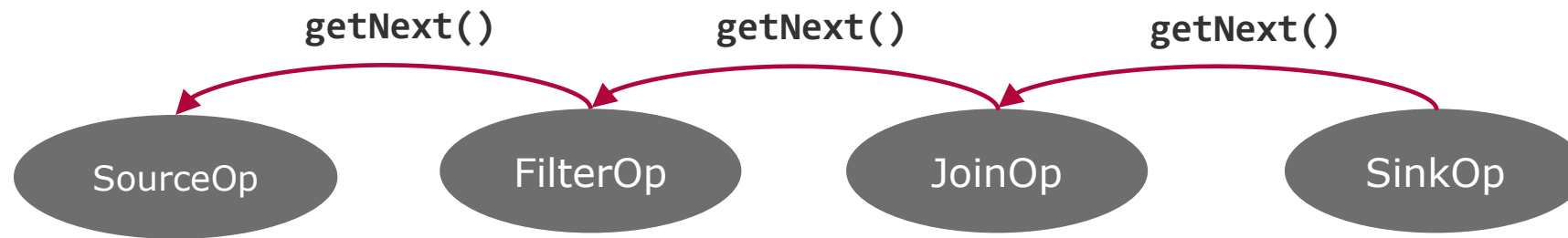
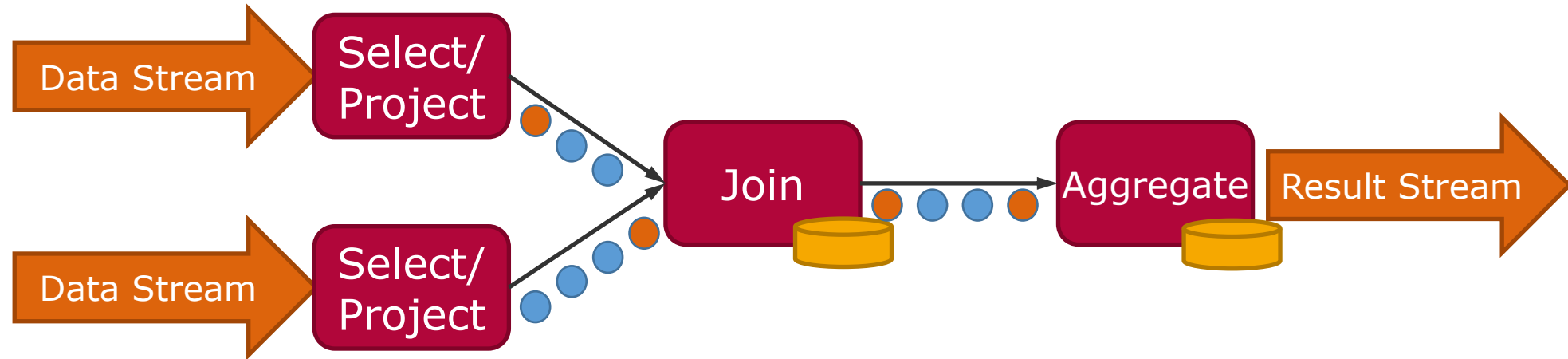
40 VMs @ \$5760/day

52-core instance @ \$6/h

Alibaba Post:

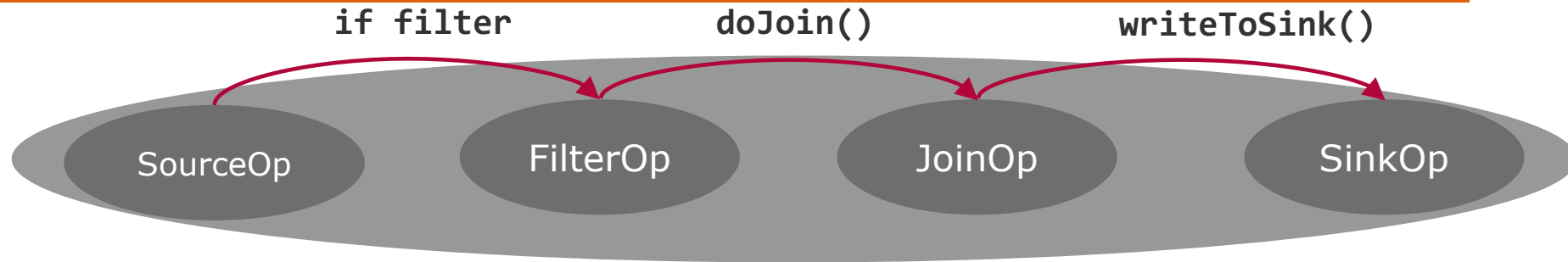
<https://www.alibabacloud.com/blog/four-billion-records-per-second-stream-batch-integration-implementation-of-alibaba-cloud-realtime-compute-for-apache-flink-during-double-11-586962>

Stream Processing – Iterator Model

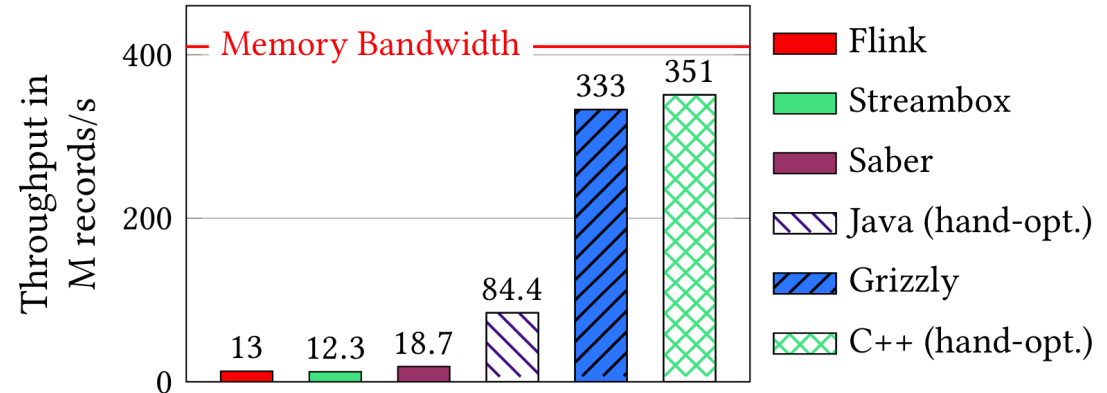
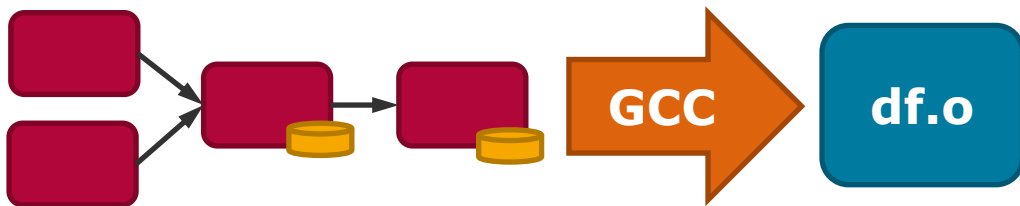


- Many virtual method calls :(
- Bad cache locality :(

Streaming Processing – Query Compilation Model



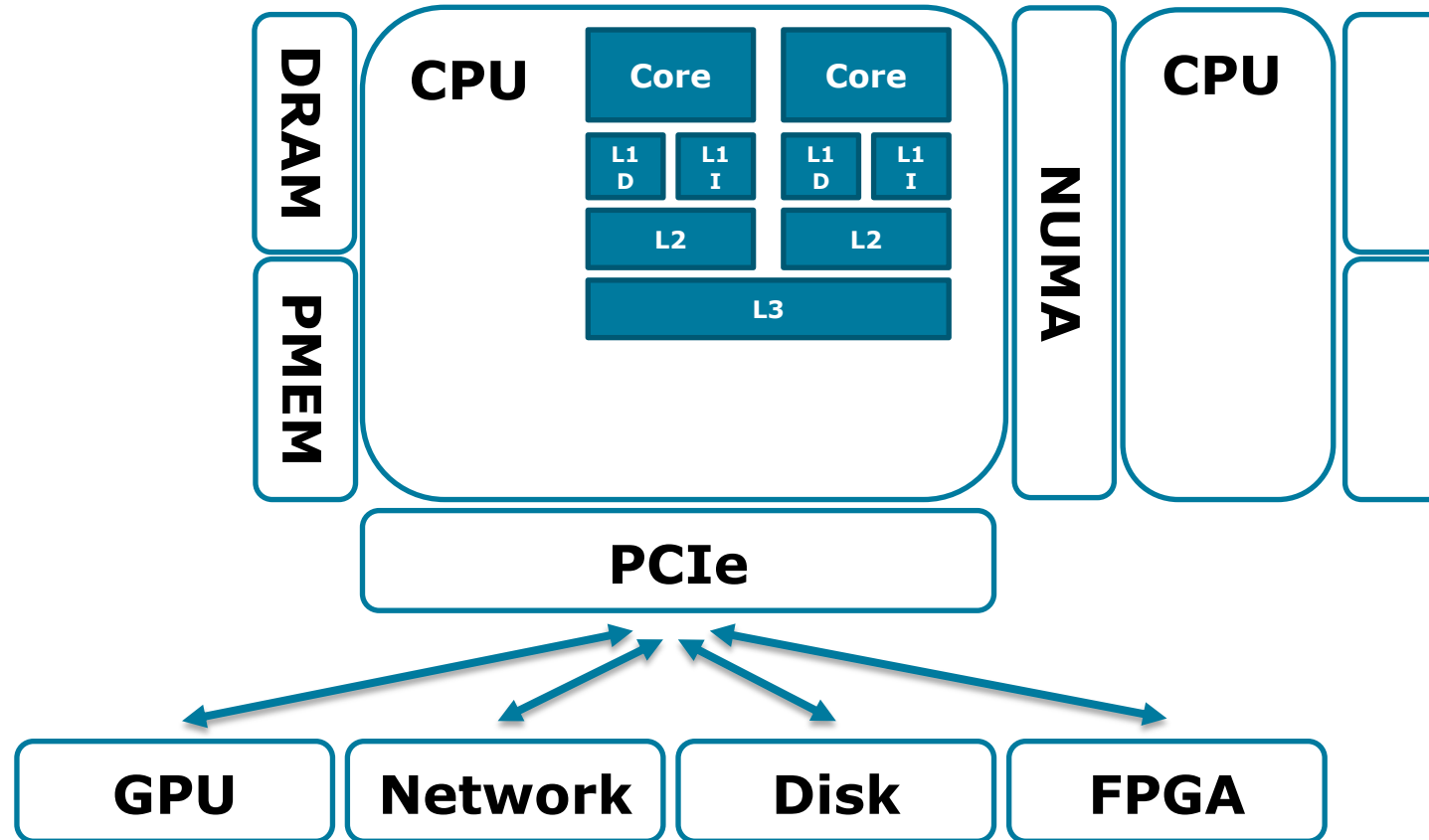
```
void pipeline() {
  for (Record rec : Source) {
    if (rec == 10) {
      doJoin(rec);
      writeToSink(joinResult);
    } } }
```



- Few/no virtual method calls :)
- Good cache locality :)

Zeuch et al. "Analyzing Efficient Stream Processing on Modern Hardware" PVLDB'19
 Grulich et al. "Grizzly: Efficient Stream Processing Through Adaptive Query Compilation" SIGMOD'20
 Benson et al. "Darwin: Scale-In Stream Processing" CIDR'22

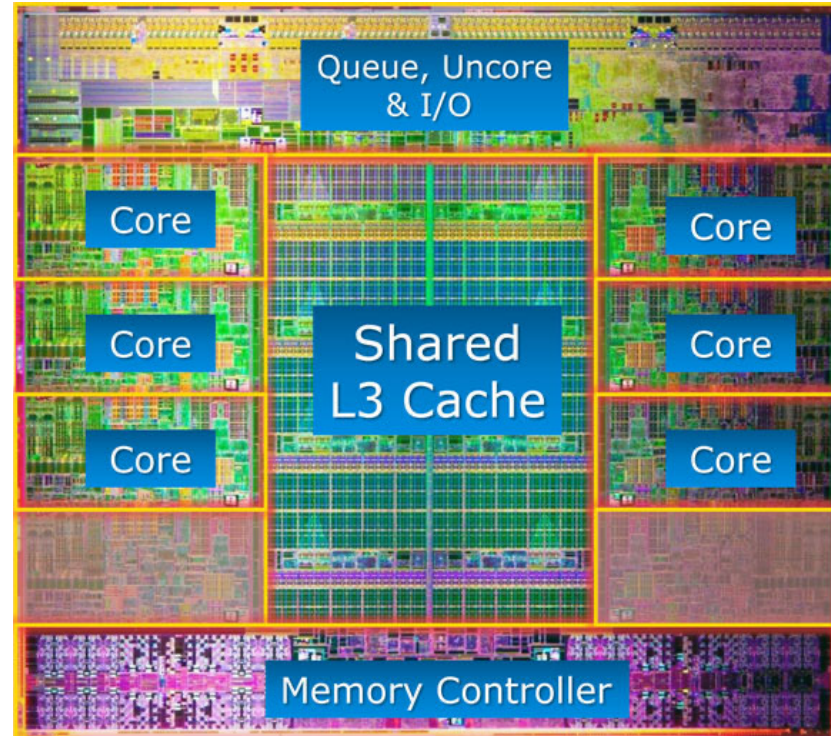
What code to generate? Let's take a closer look...



Modern computer architecture with PCIe Bus

Memory, PCIe, core and socket connection through internal bus (ring or mesh)

CPU Die – Intel i7 (from 2011)



Intel Core i7-3960X

The die is 21 by 21 mm and has 2.27 billion transistors

CPU Die II – M1 (2020)

Performance Cores

L1 Instruction Cache 192 KB

L1 Data Cache 128KB

Shared L2 Cache 12 MB

Efficiency Cores

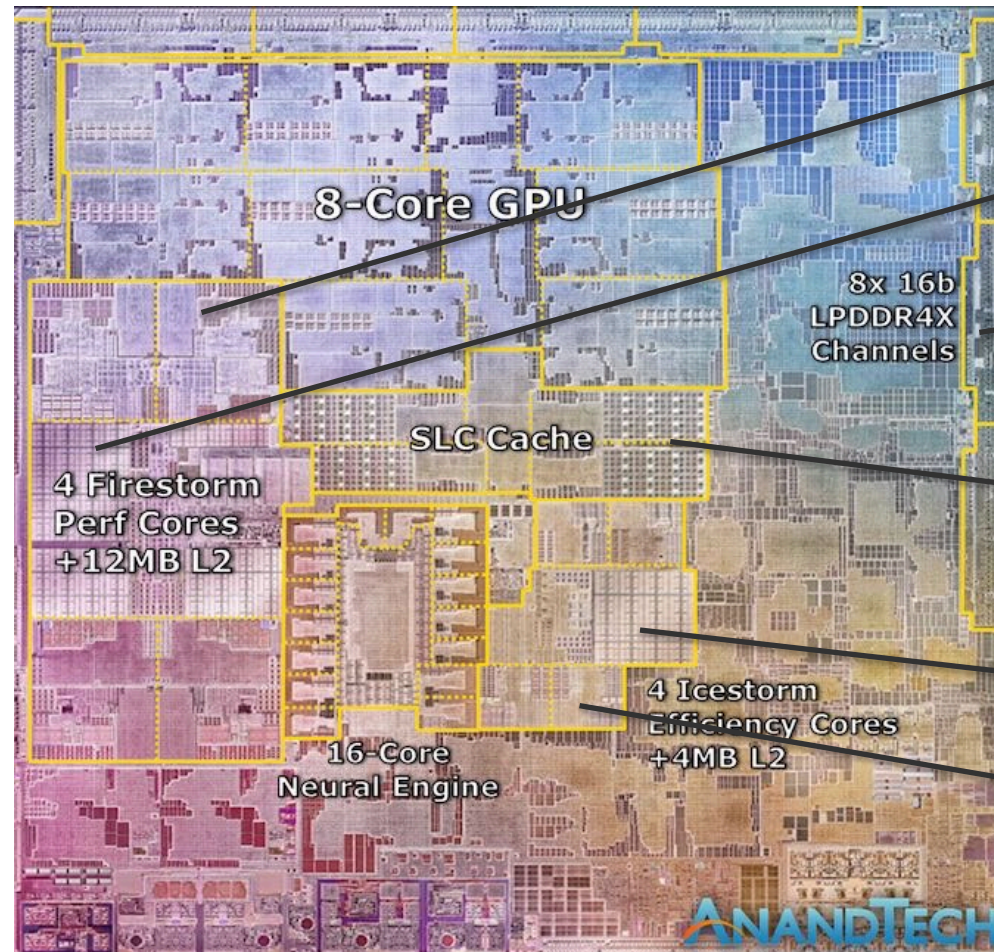
L1 Instruction Cache 128 KB

L1 Data Cache 64KB

Shared L2 Cache 4 MB

System Level Cache

Shared with GPU: 16MB



Perf Core

Perf L2

DRAM Channel

System Cache

Eff L2

Eff Core

M1 Firestorm Core

192KB L1I\$

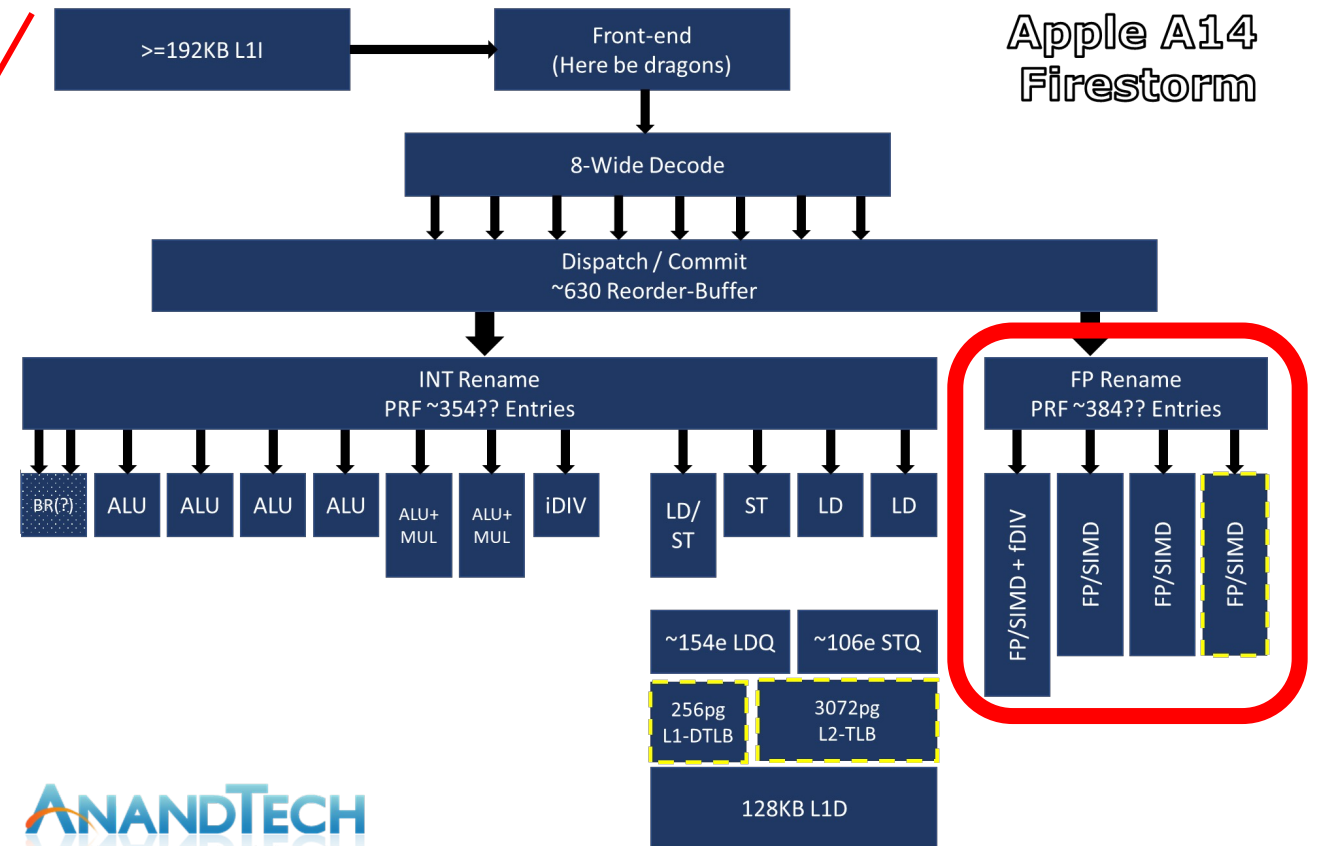
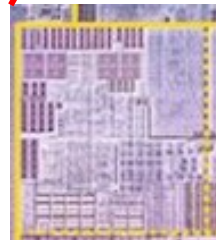
8-wide decoder

- 8 instructions per cycle

630 size reorder buffer

7 integer ports (plus ld/st,fp)

8 μ -ops per cycle



Apple A14 Firestorm

SIMD Compiler Intrinsic

Lawrence Benson, Richard Ebeling, Tilmann Rabl
ADMS 2023

SIMD in a Nutshell

Single Instruction Multiple Data (= SIMD)

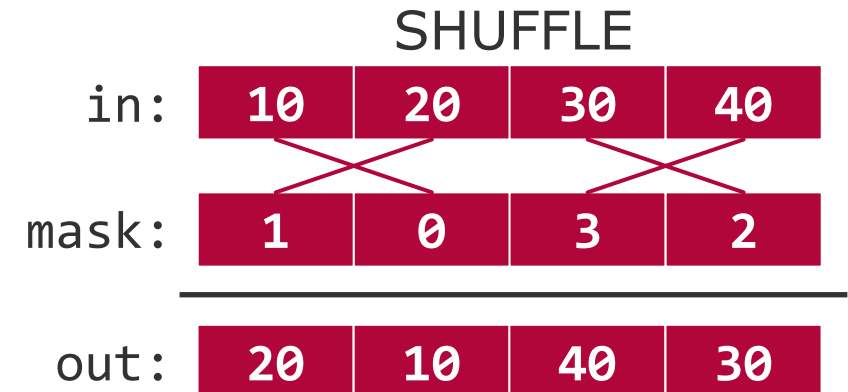
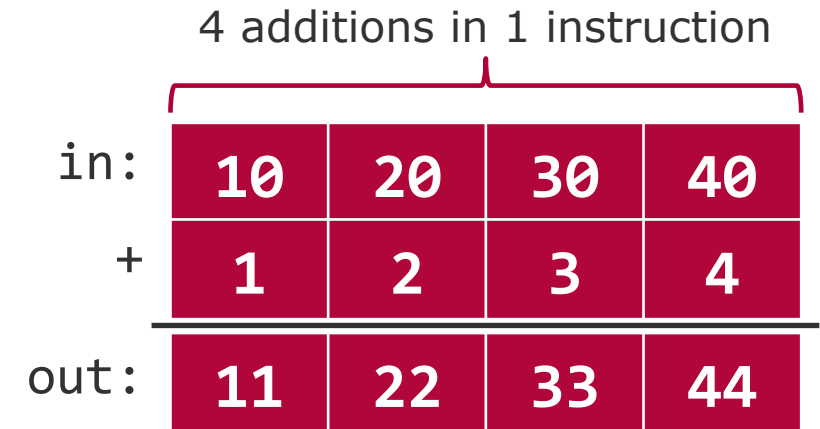
Most common instruction sets

- x86: SSE, AVX, AVX2, AVX512 (> 6k instructions)
- ARM: Neon (> 4k instructions), SVE
- PowerPC AltiVec, RISC-V V

Arithmetic, Logical, Shuffle, Shift, Load, Store, ...

Focus on x86 and Neon

- x86: 128 – 512 Bit registers
- Neon: 128 Bit registers



SIMD in Databases

Used to speed up, e.g.,

- Table scans
- Hash tables
- Sorting

SIMD code is ...

- ... hard to develop
- ... hard to test
- ... hard to benchmark

Non-x86 CPUs on the rise

- How to translate x86 SIMD code?

Rethinking SIMD Vectorization for In-Memory Databases

Orestis Polyzos
Columbia University
orestis@cs.columbia.edu

ABSTRACT
Analytical databases are underpinned by underlying hardware in order to achieve high performance. At the same time, there are many directions to explore different hardware designs. One such example, this paper designs a new SIMD architecture that relies on a Data Accessability...

SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units

Thomas Willhalm
Nicolae Popovici

Yazan Boshmaf

Hasso Plattner
Alexander Zeier

The FastLanes Compression Layout: Decoding >100 Billion Integers per Second with Scalar Code

Analyzing Vectorized Hash Tables Across CPU Architectures

Maximilian Böther
ETH Zurich
Zurich, Switzerland
mboether@inf.ethz.ch

Lawrence Benson
Hasso Plattner Institute
Potsdam, Germany
lawrence.benson@hpi.de

Ana Klimovic
ETH Zurich
Zurich, Switzerland
aklimovic@ethz.ch

Tilmann Rabl
Hasso Plattner Institute
Potsdam, Germany
tilmann.rabl@hpi.de

ABSTRACT
Data processing systems often leverage vector instructions to achieve higher performance. When applying vector instructions, an often overlooked data structure is the hash table, even though it is fundamental in data processing systems for operations such as indexing, aggregating, and joining. In this paper, we characterize and evaluate three fundamental vectorized hashing schemes, vectorized linear probing (VLP), vectorized fingerprinting (VFP), and bucket-based comparison (BBC). We implement these hashing schemes on the x86, ARM, and Power CPU architectures, as modern database engines...

Figure 1: Lookup performance of best scalar (RH) and vectorized (BBC) hashing schemes for load factor 90%.

What do these functions do?

```

_mm_add_epi32()
mm512_srl_epi64()
vaddq_s32()
    
```

Don't have **AVX512**?
→ Can't compile

Don't have **NEON**?
→ Can't compile

Abstractions on top of Abstractions

Add two 128-bit registers of 4x 32-bit integers

A	10	20	30	40
+ B	1	2	3	4
C:	11	22	33	44

Application code
vec C = A + B;

SIMD library
**struct vec {
 vec operator+(vec, vec);
}**

SIMD intrinsics
mm_add_epi32
vaddq_s32

Compiler representation
**__attribute__((
 vector_size(N)));**

Cut the Middle Man: GCC Vector Extensions

Compilers map platform-specific types and operations to canonical internal representation anyway

- Decouples from external specifications
- Allows reusing analysis and optimization steps

→ We could directly target this canonical representation with our code

- Let the compiler do platform-specific instruction selection

GCC-API allows natural code using arithmetic ($*$, $+$, $-$, $/$) and comparison ($>$, $>=$, $==$, $<$, $<=$) operators

GCC Vector Extensions (2)

```
#include <stdint>
using VecT __attribute__((vector_size(16))) = int32_t;

void add(VecT* X, VecT* Y, VecT* Z, int MAX) {
    for (int i = 0; i < MAX/4; i++) {
        *Z++ = *X++ + *Y++;
    }
}
```

Produces identical code to platform-specific intrinsics ([ARM](#), [x86](#))

```
#include <immintrin.h>

void add(__m128i* X, __m128i* Y, __m128i* Z, int MAX) {
    for (int i = 0; i < MAX/4; i++) {
        *Z++ = _mm_add_epi32(*X++, *Y++);
    }
}
```

```
#include <arm_neon.h>

void add(int32x4_t* X, int32x4_t* Y, int32x4_t* Z, int MAX) {
    for (int i = 0; i < MAX/4; i++) {
        *Z++ = vaddq_s32(*X++, *Y++);
    }
}
```

Benchmarks

M1: Apple Macbook Pro 14" M1 2021

x86 Icelake: Intel Xeon Platinum 8352Y

All experiments with:

- Clang 15 (x86) and trunk Clang ~17 (ARM)
- -O3, -march/-mtune=native

Single-threaded

Show relative speedup over scalar version

Also other x86/ARM CPUs → see paper

Bit-Packed Integer Decompression

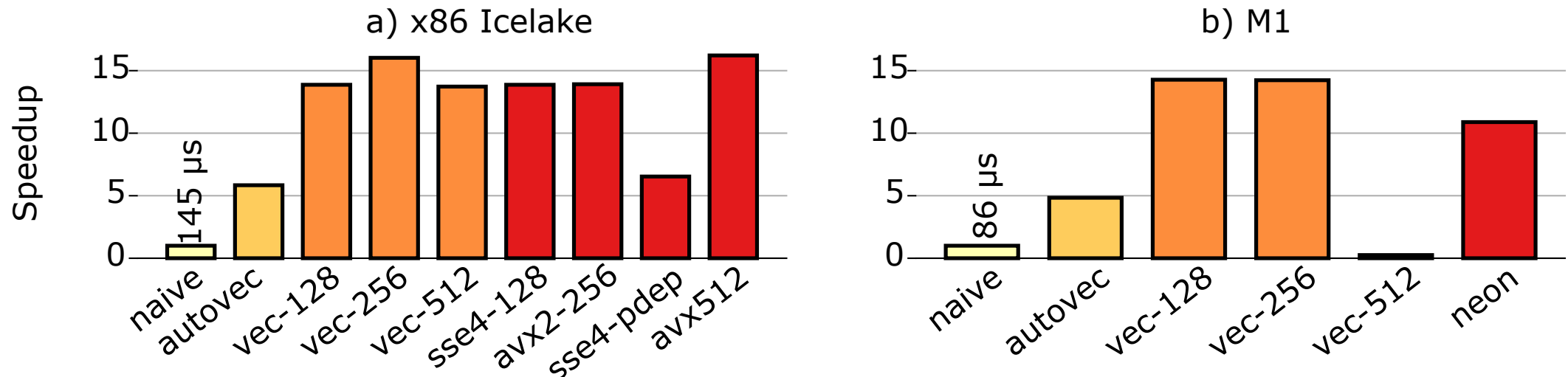
Based on VLDB '09 SIMD-Scan paper

Unpack packed 9-Bit integers to 32 bits → shuffling + shifting

Compiler **vec** ≥ **hand-written SIMD**

Neon: vec-128 generates better code than translated x86 intrinsics

- Implementation artifact on NEON



Compiler-Intrinsics in Velox

Velox: Meta's new unified query engine

Removed xSIMD dependency

Use only compiler-intrinsics

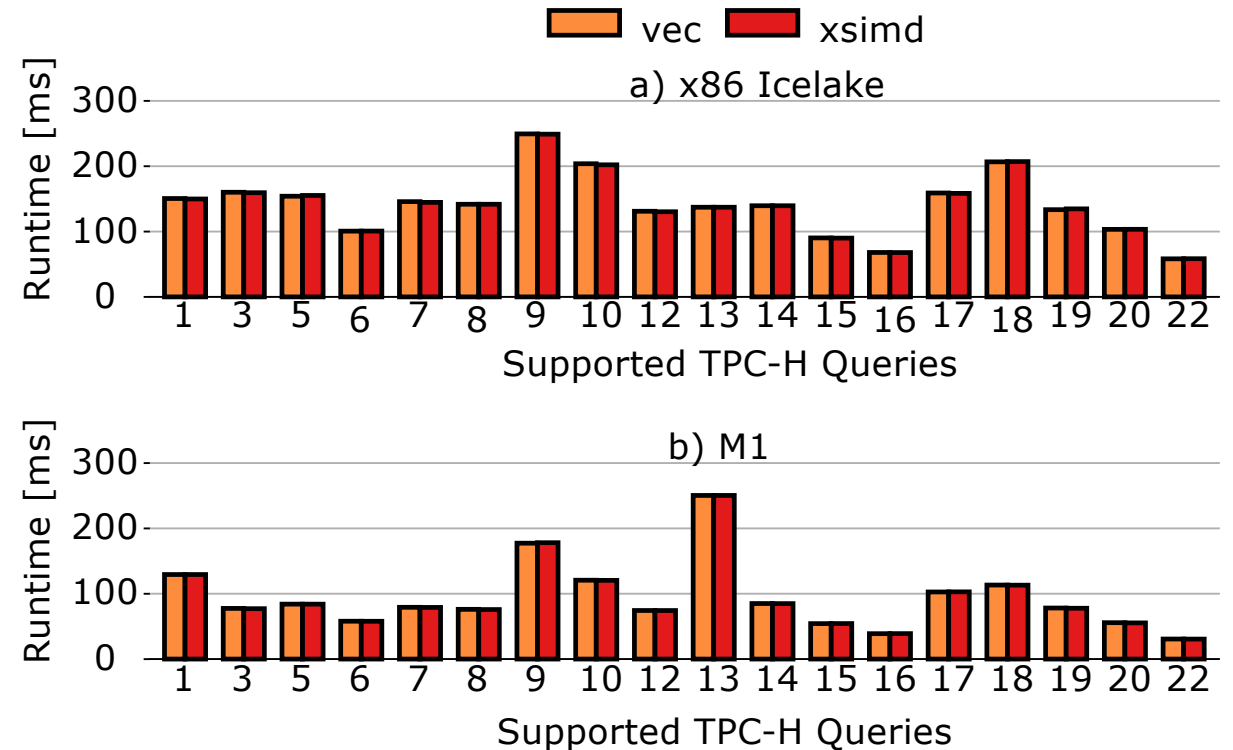
End-to-end TPC-H SF1

x86 → 0.1% diff

NEON → 0.13% diff

Removed:

- 54 platform-specific functions
- Hundreds of lines of SIMD code



A large decorative graphic element is positioned in the lower half of the slide. It features a wide, horizontal bar with a yellow-to-orange gradient. Below this bar is a solid maroon rectangular area. The word 'Outlook' is centered in white text within the maroon area.

Outlook

The Larger Picture

Efficiency is needed more than ever

- 1xChatGPT ~ 1000xGoogle¹

Efficiency does not mean *newer bigger faster*

- The cloud won't solve this
- There is little economic incentive in being most efficient

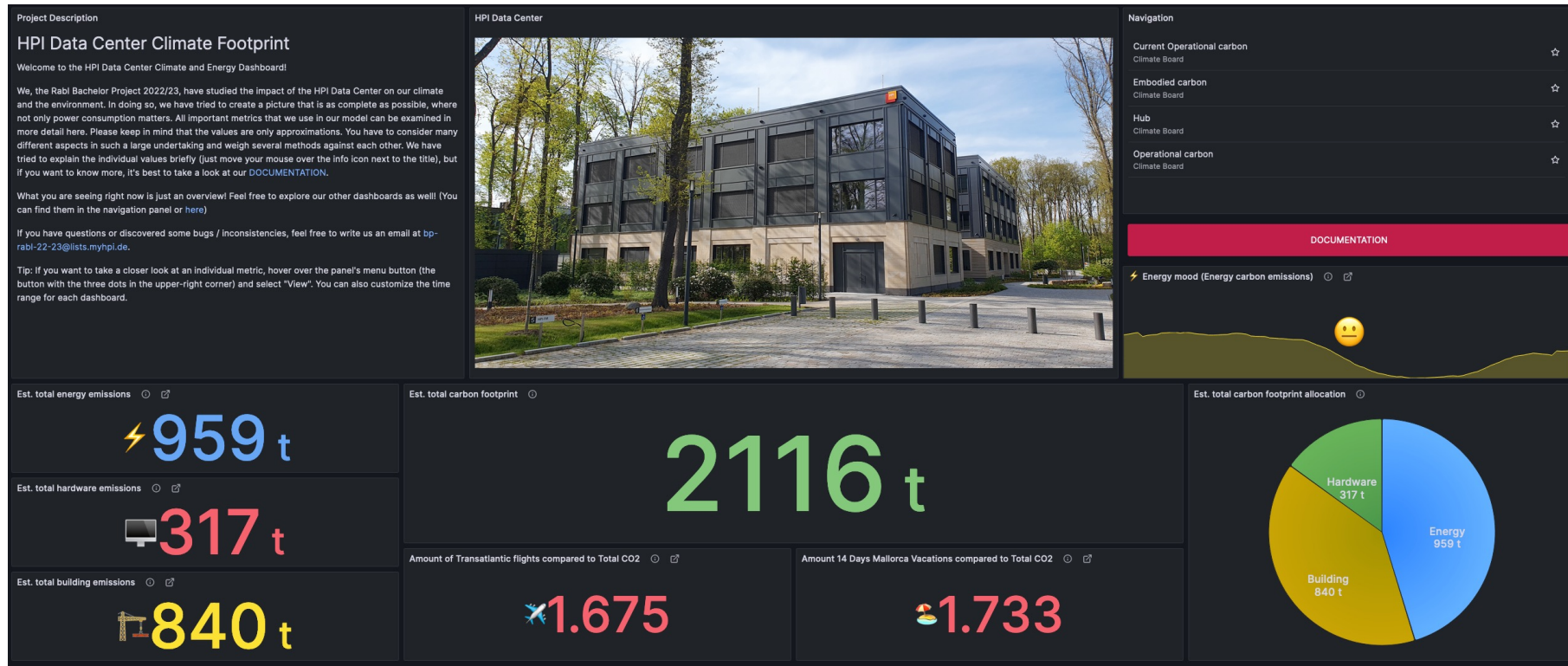
Need to make best use of what you have



<https://www.tagesschau.de/multimedia/bilder/blickpunkte-8196.html>

¹<https://ai.stackexchange.com/questions/38970/how-much-energy-consumption-is-involved-in-chat-gpt-responses-being-generated>

Measuring Carbon Footprint



<https://climateboard-bptr1.hpi.de>

Thank you for your attention!

- Course Motivation and Contents
- Course Logistics
- Stream Processing

- Next session: Pinar Tözün – 24.10.

24.10.



Pinar Tözün

ITU Copenhagen

Hardware Parallelism &
Transaction Processing Systems