Excalibur

A Virtual Machine for Adaptive Fine-grained JIT-Compiled Query Execution based on VOILA

Tim Gubner*

Peter Boncz

*Now at *Huawei Cloud Norway*, research was done at CWI

CWI
Database Architectures

Background

A query has multiple possible implementations (“flavors”) ⇨ “Design Space”

- Vectorized, data-centric, etc.
- With different performance characteristics[1], depending on hardware[2]
- With increasingly heterogeneous hardware, likely to get worse

[1]Kersten et al. Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask. VLDB 2018

[2]Gubner, Boncz. Highlighting the Performance Diversity of Analytical Queries using VOILA. ADMS 2021

VOILA Recap

(continuation of work on VOILA[1])

- VOILA = Domain-Specific Language
- Generate different static implementations from *one* description
  - e.g. data-centric, vectorized, SIMD …

[1]Gubner, Boncz. Charting the Design Space of Query Execution using VOILA. VLDB 2021.

Challenge(s)

1. How can a query engine exploit the design space?
- Systems typically tied to *one* flavor (e.g. vectorized)
- Design space hardly predictable: online learning (adaptivity)

2. Can we explore more efficiently?
- VOILA used random sampling (not great?)
- Consequence of adaptivity, due to limited exploration time
  - Max. benefit, is at start of query (“asap”)
  - Excessive exploration will hurt performance

Excalibur

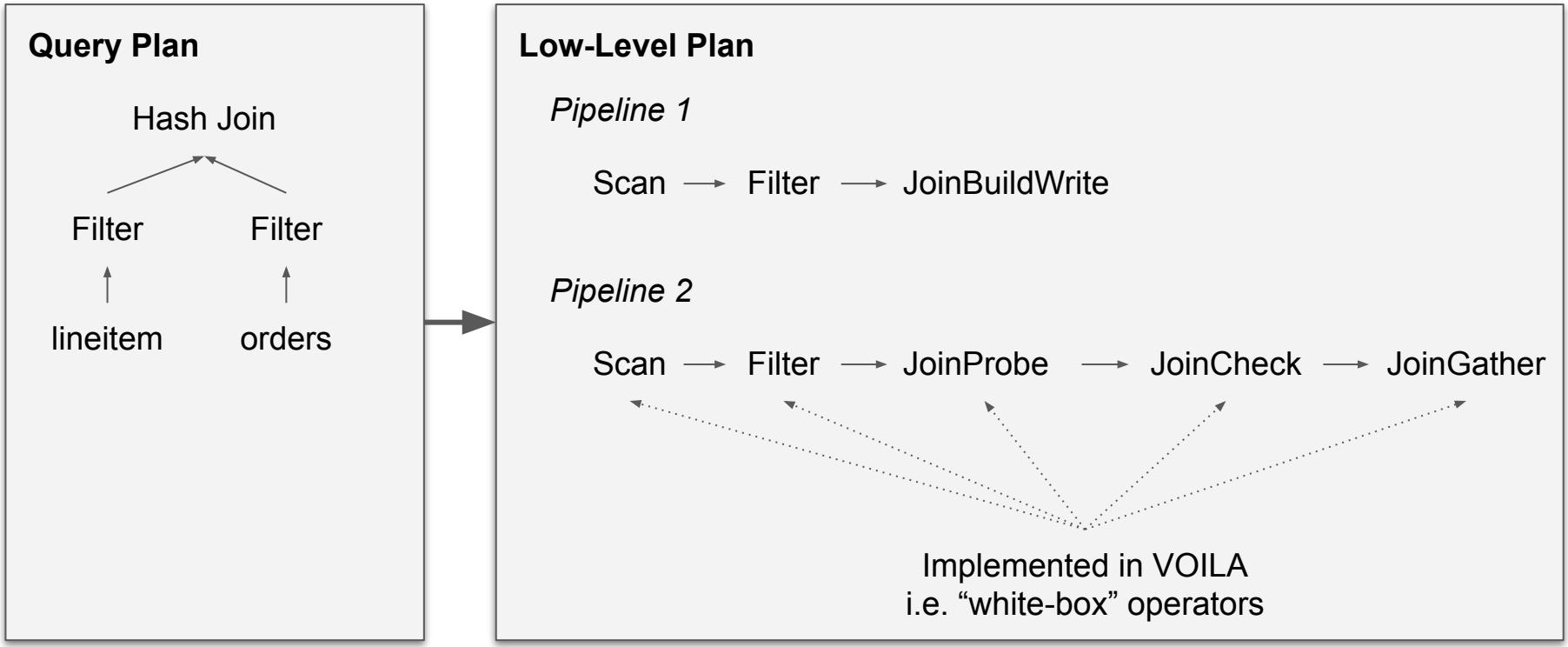
- JIT-compiling (vectorized) engine

- Exploring the design space is **risky**, no guarantee in finding good points

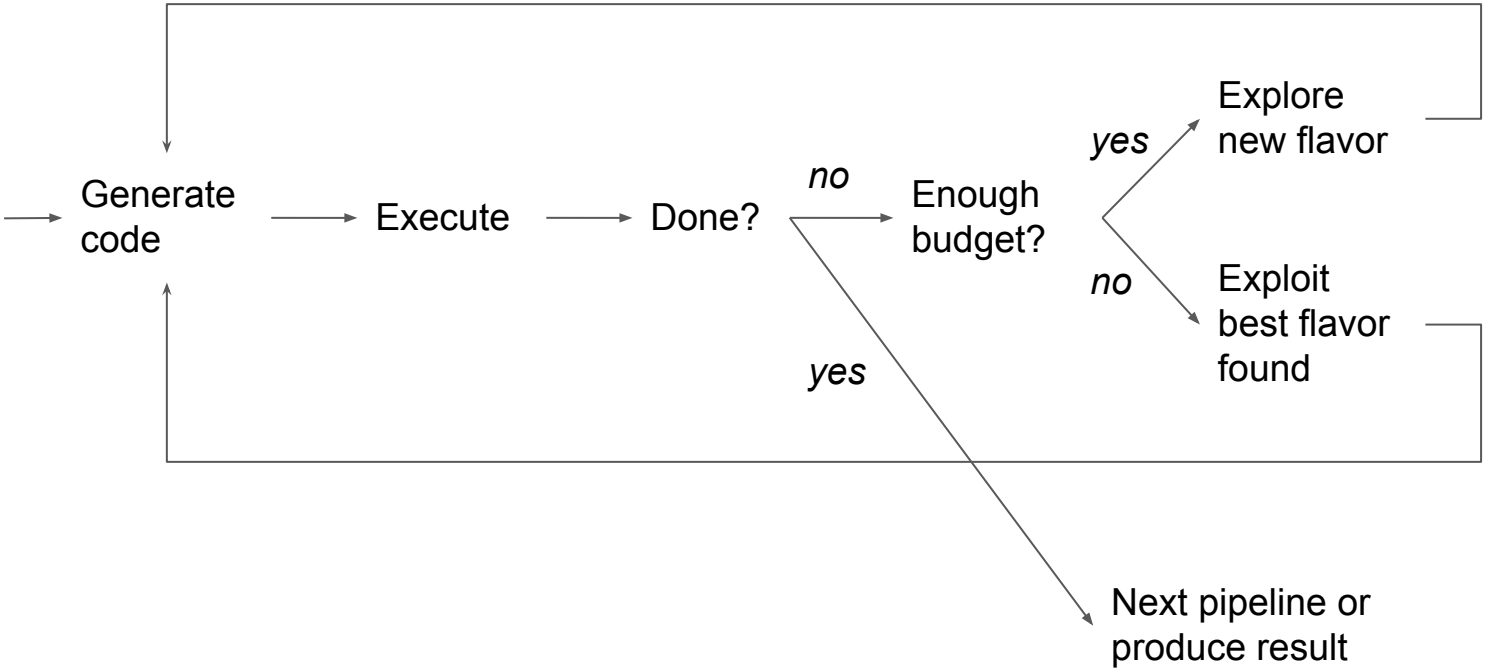
Primum non nocere = first, do no harm

- First execute using vectorized interpretation
- Then try to improve:
  - First explore design space
  - Exploit fastest point found

A Query in Excalibur



A Pipeline in Excalibur



Generating Code

1. Apply high-level rewrite rules
2. Select VOILA fragments
3. Generate code for fragments, or *reuse cached fragments*
4. Generate byte code with calls to fragments

Rather generic ⇨ *Many possibilities!*

Generating Code

1. Apply high-level rewrite rules
2. Select VOILA fragments
3. Generate code for fragments, or *reuse cached fragments*
4. Generate byte code with calls to fragments

1. Inline operators into last op.
2. Whole Pipeline = Fragment

Data-Centric Execution

1. Intro. Bloom filter

Bloom filter + Vectorized Execution

1. Reorder ops

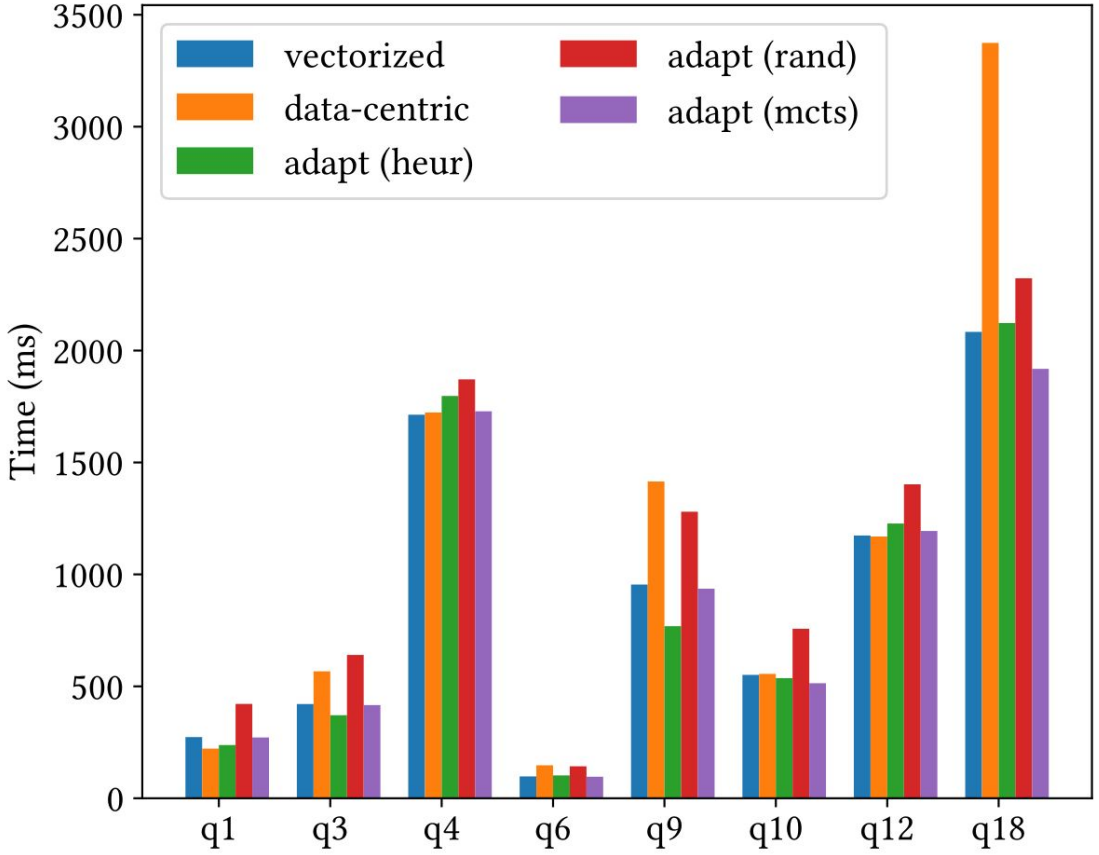
Diff. filter order + Vectorized Execution

Excalibur vs. Analytical Systems

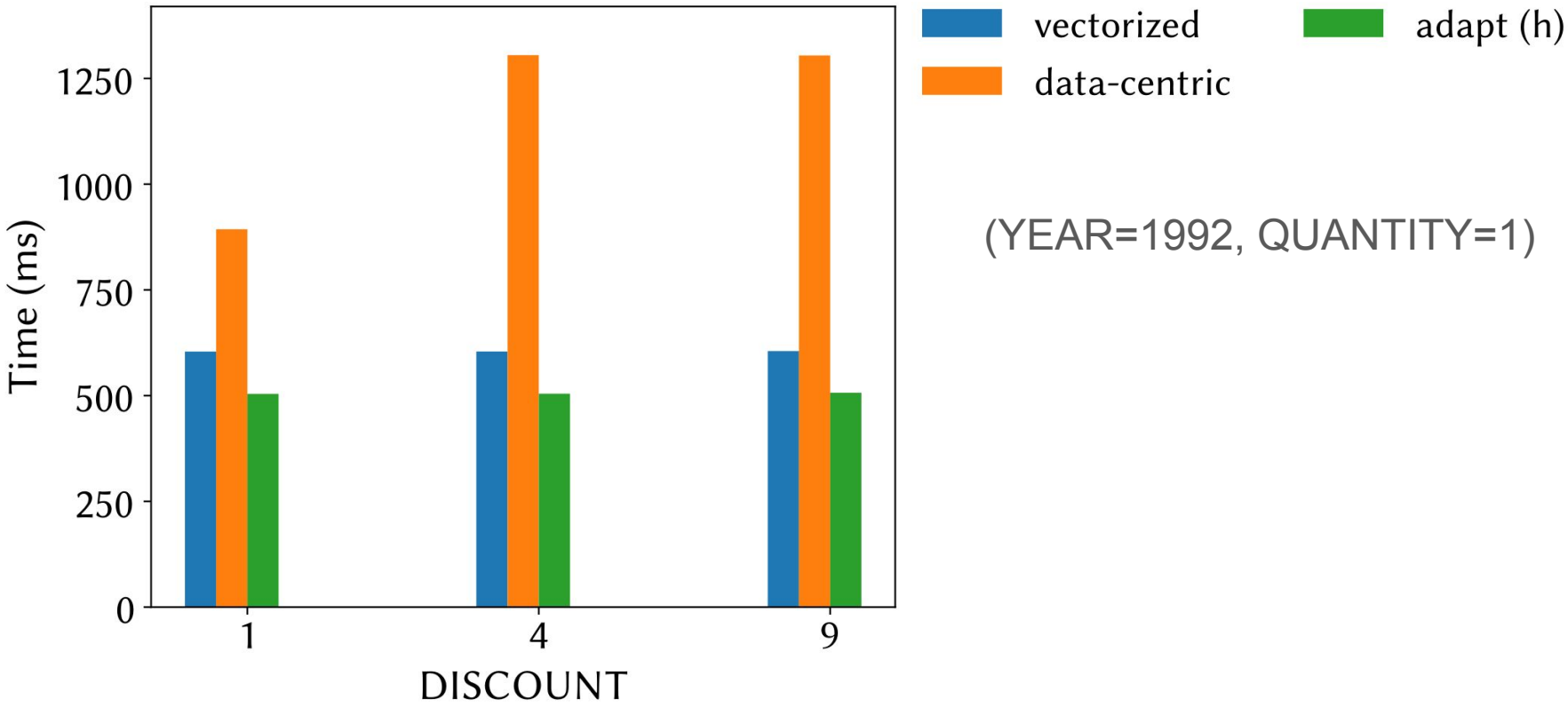
	Q1	*Q3*	*Q6*	*Q9*
Umbra	287	**326**	91	854
DuckDB	1325	2338	341	15306
MonetDB	5488	1089	190	1178
Excalibur (heur)	**192**	349	**52**	**730**

- Runtime in ms
- Multi-threaded on TPC-H SF50

Execution Tactics on TPC-H SF 50



Adaptivity in TPC-H Q6



Summary

- No need to make static choices anymore!
- Have multiple flavors in Excalibur, and let the system take over

- Paper:
  - Utilize DSL to generate different flavors (VOILA)
  - Adaptivity in JIT-compiled system
  - Finding good execution tactics

- Open Source: https://github.com/t1mm3/db_excalibur