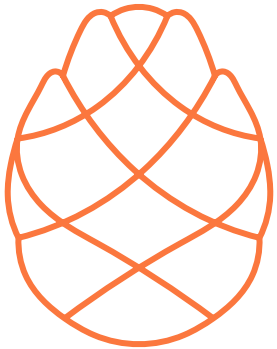


CedarDB

Philipp Fent
philipp@cedardb.com

Overview

- “PostgreSQL for analytics”
  - Full-featured versatile database
  - Simultaneous high-performance analytics and operations on the same data
  - **Several orders of magnitude** speedup over existing systems
  - Full utilization of modern hardware capabilities (e.g. massive parallelism, RAM capacity)
  - Transparently and gracefully scales **beyond main memory**

- Started at TUM
  - 5 PhDs, developed the system over the last ~6 years
  - Bring the most efficient data processing engine to the world

Database Systems

- Who uses a database system daily?

CedarDB

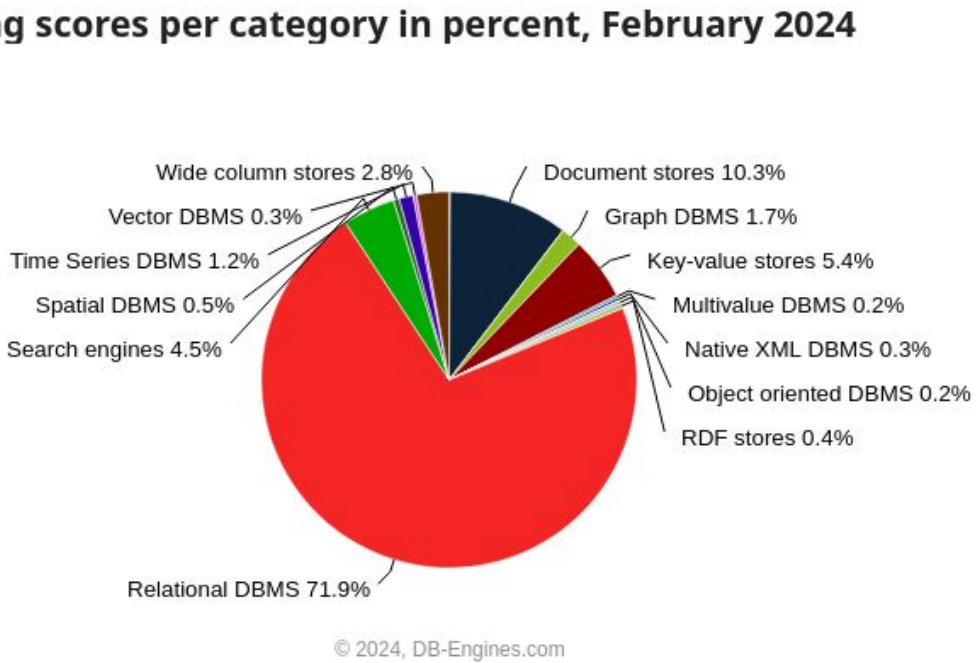
Database Systems

- Who uses a database system daily?
- SQL or NoSQL?

CedarDB

Database Systems

- Who uses a database system daily?
- SQL or NoSQL?



SQL / Relational

- Old

NoSQL

- Not so old

CedarDB

SQL / Relational

- ~~Old~~
- Ancient

NoSQL

- Not so old

CedarDB

SQL / Relational

- ~~Old~~
- Ancient

NoSQL

- Not so old

SQL / Relational

- ~~Old~~
- Ancient

NoSQL

- Not so old
- High scalability
- Flexible data formats
- Simple key/value storage

SQL / Relational

- ~~Old~~
- Ancient

- But an efficient model to work with data
- Decades of experience building data processing pipelines for data-driven applications

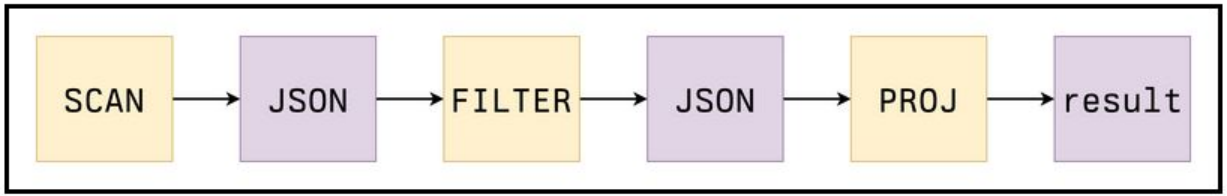
NoSQL

- Not so old
- High scalability
- Flexible data formats
- Simple key/value storage

Case Study: MongoDB

Case Study: MongoDB

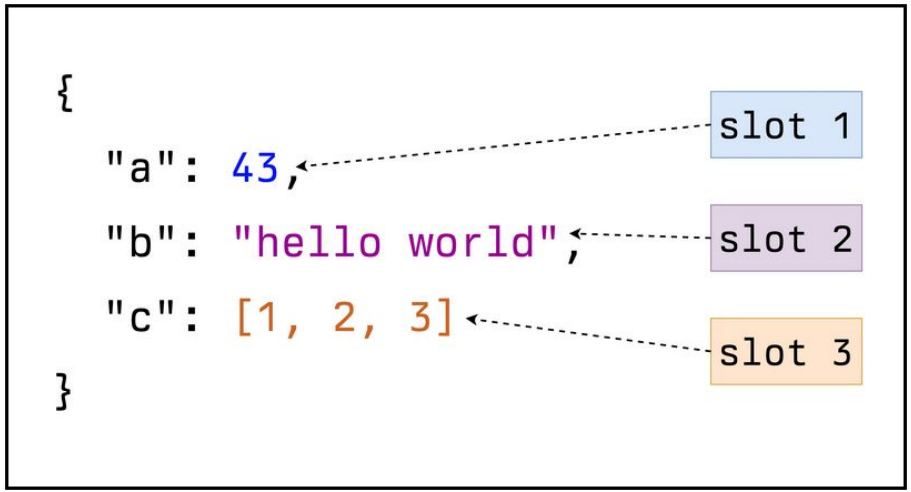
- New query engine in MongoDB 6 (great write-up on laplab.me)
- Previous MongoDB engine was document oriented



> *A couple of intermediate query stages, each creating a JSON document to pass to the next one. All documents are discarded except the last one on the right.*

Case Study: MongoDB

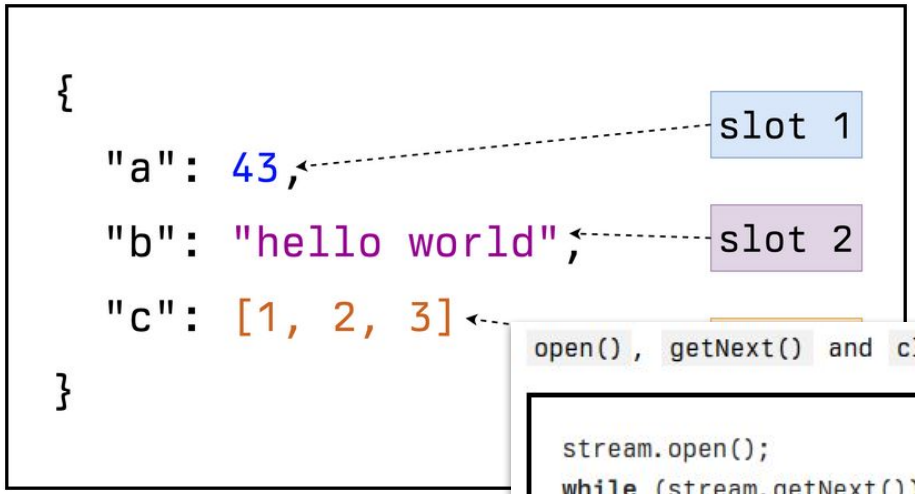
- Slot-Based Query Execution Engine



CedarDB

Case Study: MongoDB

- Slot-Based Query Execution Engine



`open()`, `getNext()` and `close()` interface and can be used like this:

```
stream.open();
while (stream.getNext()) {
    // Do something with the data stream provided.
}
stream.close();
```

CedarDB

Case Study: MongoDB

- Volcano model query processing

120 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 6, NO. 1, FEBRUARY 1994

Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

***Abstract*—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.**

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using *support functions*. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is *extensible* with new operators, algorithms, data types, and type-specific methods.

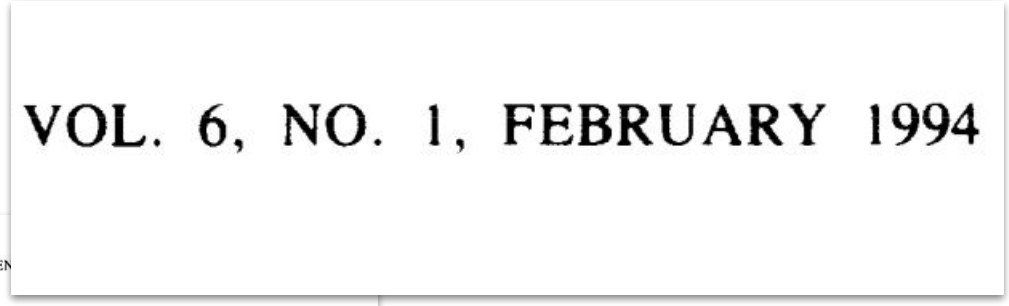
Volcano includes two novel *meta-operators*. The *choose-plan* meta-operator supports *dynamic query evaluation plans* that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The *exchange* meta-operator supports *intra-operator parallelism* on parti-

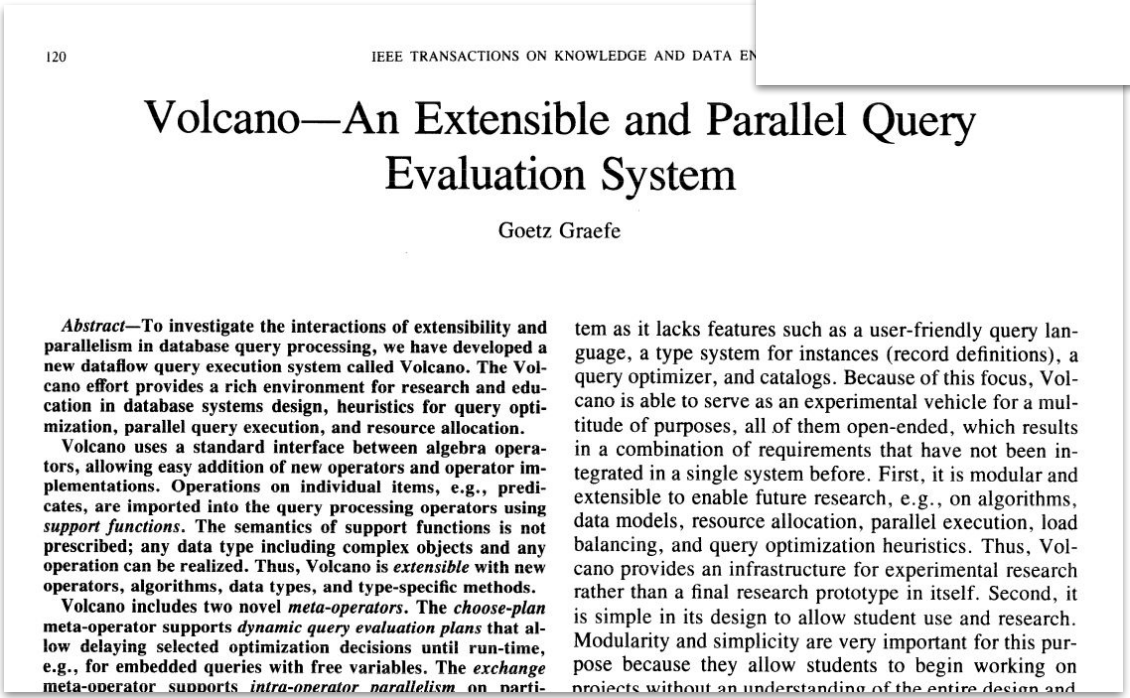
tem as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it is simple in its design to allow student use and research. Modularity and simplicity are very important for this purpose because they allow students to begin working on projects without an understanding of the entire design and

CedarDB

Case Study: MongoDB

- Volcano model query processing





CedarDB

Case Study: MongoDB

- Volcano model query processing

VOL. 6, NO. 1, FEBRUARY 1994

Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

***Abstract*—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.**

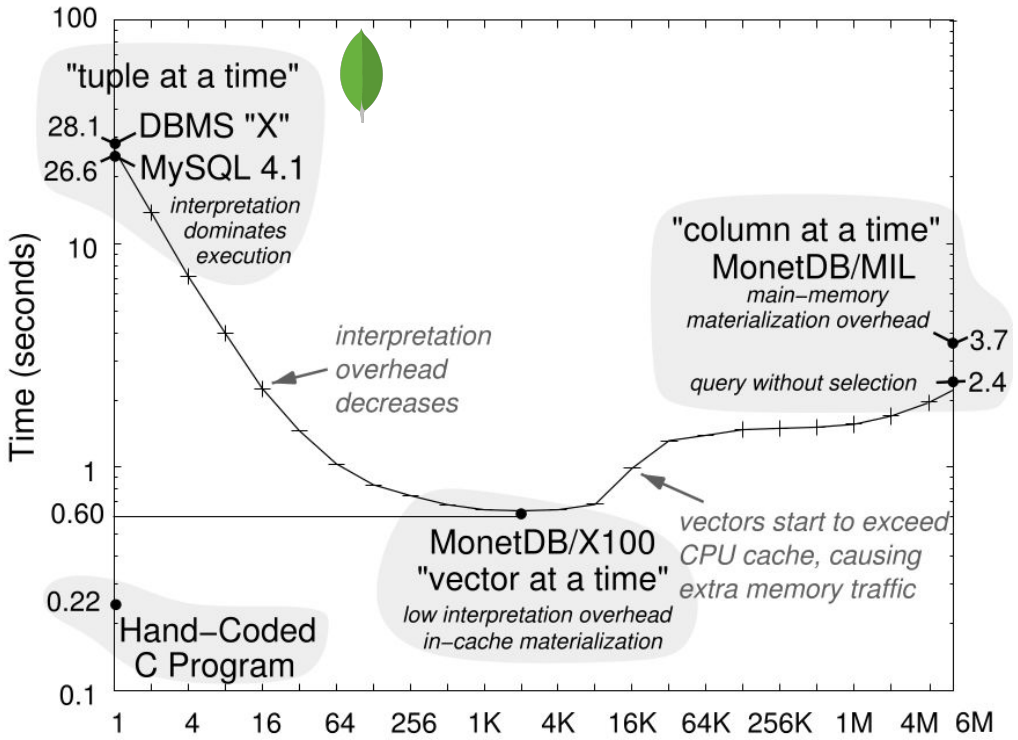
Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using *support functions*. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is *extensible* with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel *meta-operators*. The *choose-plan* meta-operator supports *dynamic query evaluation plans* that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The *exchange* meta-operator supports *intra-operator parallelism* on parti-

tem as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it is simple in its design to allow student use and research. Modularity and simplicity are very important for this purpose because they allow students to begin working on projects without an understanding of the entire design and

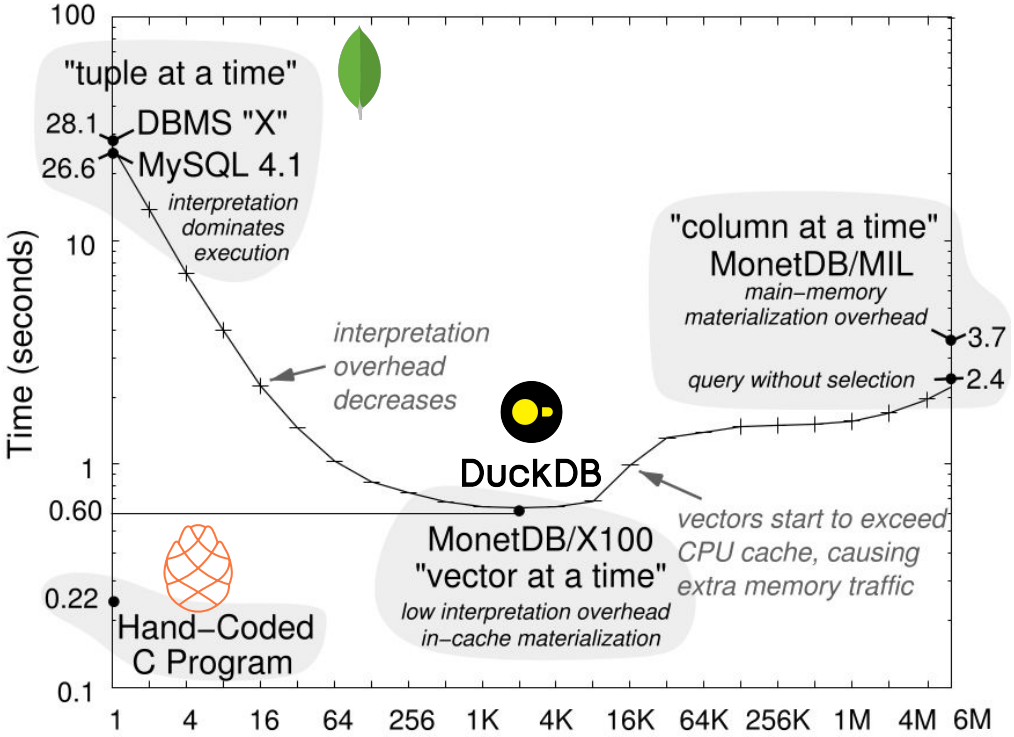
CedarDB

State of the art



CedarDB

State of the art

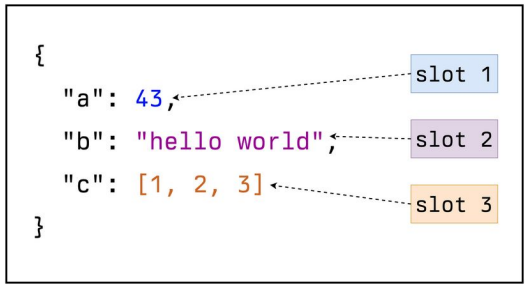


CedarDB

Code generation

- Query execution plan similar to hand-coded C
- Slots ≈ CPU registers

```
for doc in docs:
    a = json::lookup(doc, "a")
    b = json::lookup(doc, "b")
    c = json::lookup(doc, "c")
    sum[b] += a
```



CedarDB

Data-centric code generation

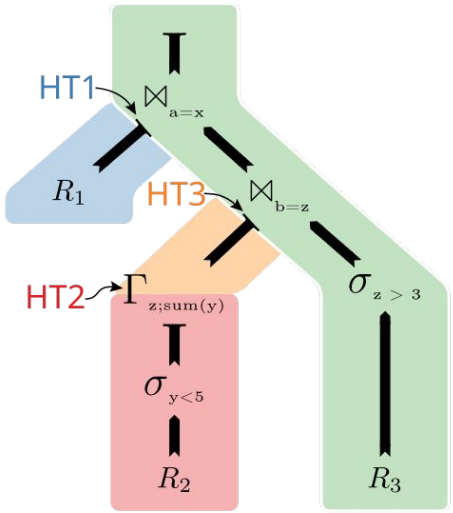
- Build data pipelines as tight loops
- Keeps data in registers as long as possible

```
for t in R1:
   HT1[t.a] = t

for t in R2:
   if t.y < 5:
      HT2[t.z] += t.y

for t in HT2:
   HT3[t.b] = t

for t in R3:
   if t.z > 3:
      if HT3[t.z]:
         if HT1[t.x]:
            print t
```



CedarDB

Expression compilation

```
%1 = zext i64 %int1;                                  Zero extend to 64 bit
%2 = zext i64 %int2;
%3 = rotr i64 %2, 32;                                          Rotate right
%v = or i64 %1, %3;                                   Combine int1 and int2
%5 = crc32 i64 6763793487589347598, %v;                         First crc32
%6 = crc32 i64 4593845798347983834, %v;                        Second crc32
%7 = rotr i64 %6, 32;                                      Shift second part
%8 = xor i64 %5, %7;                                      Combine hash parts
%hash = mul i64 %8, 11400714819323198485;                          Mix parts
```

Efficient code generation for arbitrary expressions

- Example: Efficient hashing of two 32-bit columns
- Generalizes to arbitrary data types
- And to arbitrary number of columns

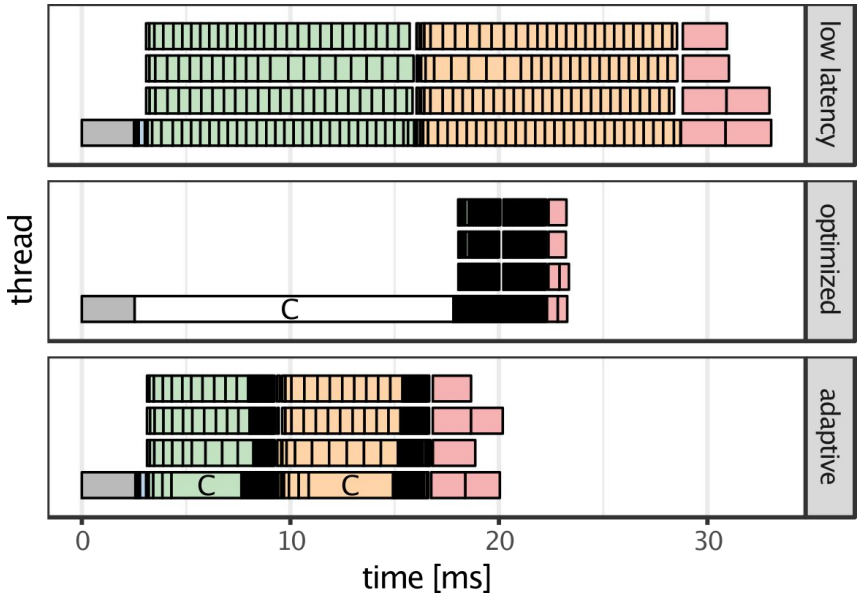
Roadmap

- ✅ Flexible data formats
- ✅ Efficient data pipelines
- ✅ Efficient execution with arbitrary complex expressions
- Scalability

CedarDB

Scalable execution

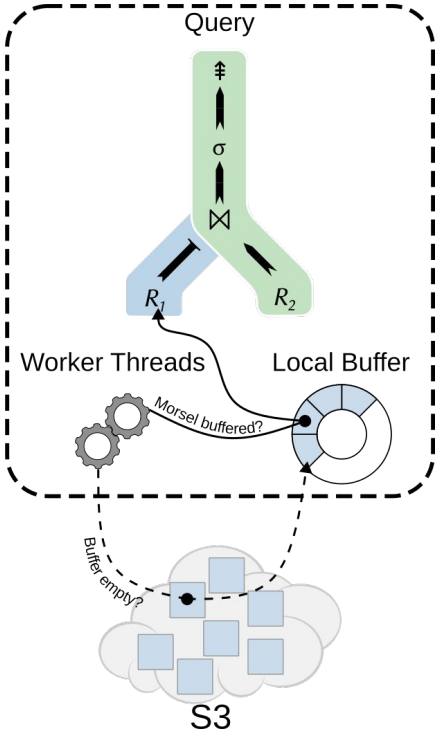
- Morsel-Driven Parallelism
  - **Full intra-query parallelism**
  - Self-adapting morsel sizes
- Scheduler
  - Adaptive scheduling of short- and long-running queries
  - **Extensible** task-based interface
- Adaptive Compilation
  - Re-compilation and re-optimization
  - Flying start: **Directly emit x86**



CedarDB

Scaling with remote storage

- Fast relations on external storage
  - **Interleaved networking** and processing
- Asynchronous networking
  - Download close to network bandwidth
  - Up to **80 Gbit/s** from S3
- S3-optimized storage layout
  - Columnar format
  - Small materialized aggregates



Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de

CedarDB

Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de


```
{
    "all_attributes":{"_registerArt":"HRB","_registerNummer":"141703",
    "additional_data":{"AD":true,"CD":true,"DK":true,"HD":false,"SI":true,"UT":true,"VÖ":false},
    "federal_state":"Bavaria",
    "native_company_number":"München HRB 141703",
    "registered_office":"Garching","registrar":"München"},
    "company_number":"D2601V_HRB141703",
    "current_status":"currently registered",
    "jurisdiction_code":"de",
    "name":"UnternehmerTUM GmbH",
    "officers":[
        {"end_date":"2010-02-11","name":"Bernward Doctor Jopen","other_attributes":{"city":"Gräfelfing","dismissed":true,"firstname":"Bernward"
        {"end_date":"2018-01-22","name":"Claudia Anke Frey","other_attributes":{"city":"Neufahrn","dismissed":true,"firstname":"Claudia Anke","
        {"name":"Andreas Doctor Liebl","other_attributes":{"city":"Unterföhring","firstname":"Andreas","lastname":"Doctor Liebl"},"position":"G
        {"name":"Claudia Anke Frey","other_attributes":{"city":"Neufahrn b. Freising","firstname":"Claudia Anke","flag":"sole representation","
        {"name":"Claudia Anke Frey","other_attributes":{"city":"Neufahrn","firstname":"Claudia Anke","lastname":"Frey"},"position":"Prokurist",
        {"name":"Helmut Doctor Schönenberger","other_attributes":{"city":"München","firstname":"Helmut","flag":"mit der Befugnis im Namen der G
        {"name":"Helmut Doctor Schönenberger","other_attributes":{"city":"München","firstname":"Helmut","flag":"sole representation","lastname"
        {"name":"Stefan Drüssler","other_attributes":{"city":"München","firstname":"Stefan","lastname":"Drüssler"},"position":"Geschäftsführer"
    ],
    "registered_address":"Lichtenbergstr.","retrieved_at":"2019-01-31T00:07:28Z"
}
```

CedarDB

Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de
- Munich’s most wanted
- Who-knows-who of Munich

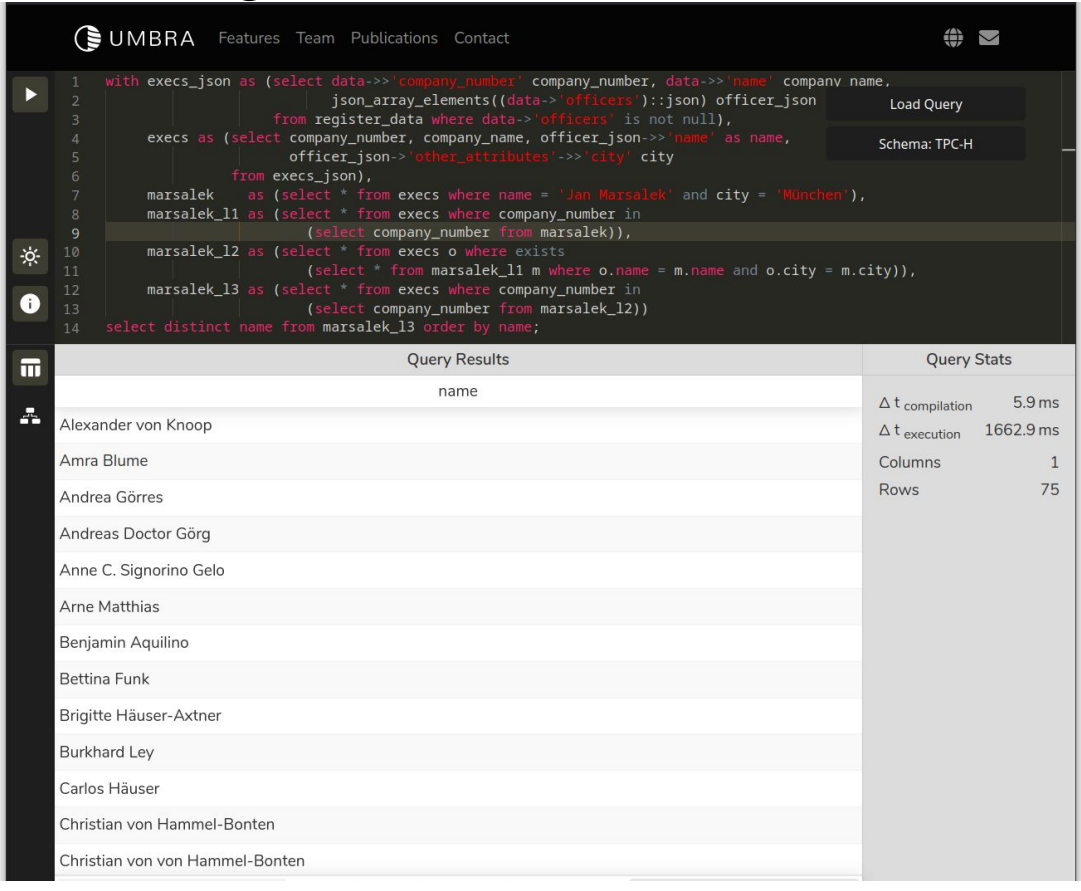
CedarDB

Six Degrees of Jan Marsalek

```
with execs_json as (select data->>'company_number' company_number, data->>'name' company_name,
                           json_array_elements((data->'officers')::json) officer_json
                    from register_data where data->'officers' is not null),
     execs as (select company_number, company_name, officer_json->>'name' as name,
                      officer_json->'other_attributes'->>'city' city
               from execs_json),
     marsalek    as (select * from execs where name = 'Jan Marsalek' and city = 'München'),
     marsalek_l1 as (select * from execs where company_number in
                        (select company_number from marsalek)),
     marsalek_l2 as (select * from execs o where exists
                        (select * from marsalek_l1 m where o.name = m.name and o.city = m.city)),
     marsalek_l3 as (select * from execs where company_number in
                        (select company_number from marsalek_l2))
select distinct name from marsalek_l3 order by name;
```

CedarDB

Six Degrees of Jan Marsalek



DBMS	Time
CedarDB	1.5s
DuckDB	13s
PostgreSQL	15s
Relational CedarDB	100ms

CedarDB

Try it now

- Full-featured versatile database
- Simultaneous high-performance analytics and operations on the same data

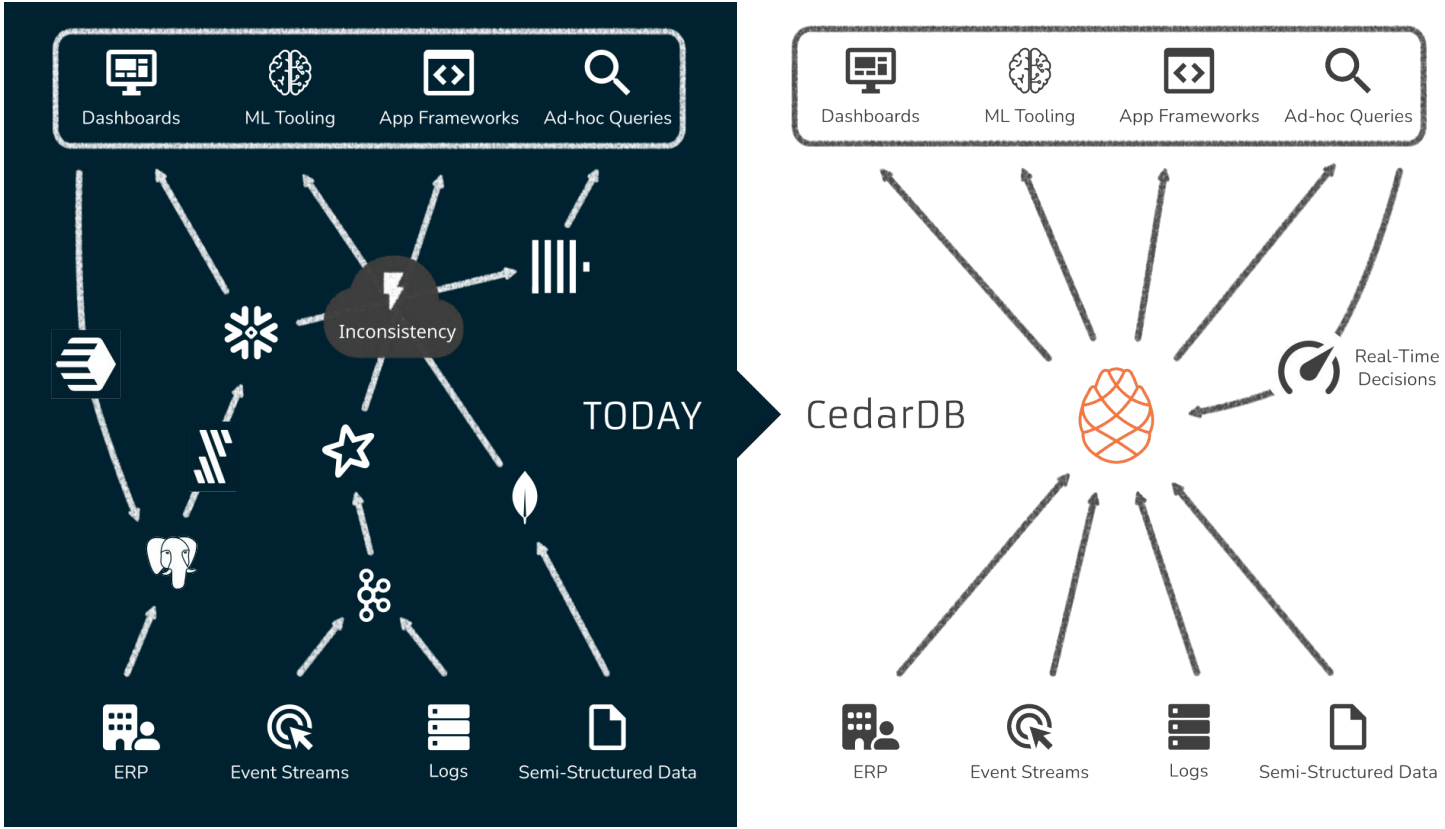
Dive deeper:

cedardb.com/docs

```
docker pull pfent/umbra
```

Get in touch:

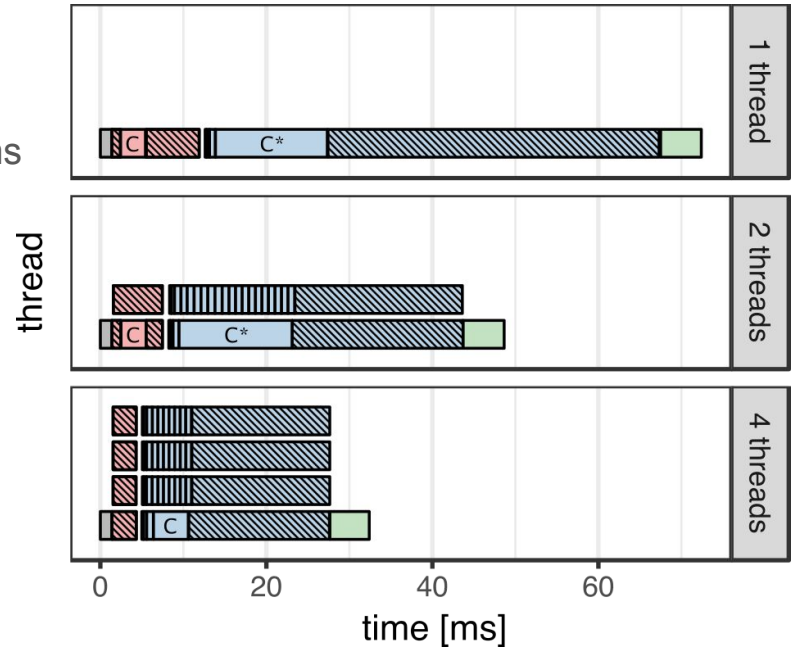
philipp@cedardb.com


Dashboards
ML Tooling
App Frameworks
Ad-hoc Queries
Inconsistency
TODAY
ERP
Event Streams
Logs
Semi-Structured Data
Dashboards
ML Tooling
App Frameworks
Ad-hoc Queries
CedarDB
Real-Time
Decisions
ERP
Event Streams
Logs
Semi-Structured Data

CedarDB

Adaptive Compilation

- Multiple JIT Backends
  - **High Efficiency** vs. **Low Latency**
  - Modularized for different requirements & platforms
- Adaptive Query Execution
  - Problem: Selecting strategy upfront is hard
  - Solution: **Start quickly** & **upgrade later**
  - Robust decisions with runtime feedback
  - Worker threads don't idle during single-threaded LLVM optimisations



CedarDB

Scheduling

- Low latencies under high load
  - Compute burned on heavy queries
  - Finish light queries quickly
  - Example: *95% system load, 75% light + 25% heavy*

    Light queries almost not affected by load.
- Adaptive morsel sizes
  - Fairness through normalized time slices
  - Simplifies adaptive compilation