Morel: A language for data

Julian Hyde

South Bay Systems meetup · Mountain View, CA · November 19, 2025

South Bay Systems: Morel / Query Optimization as a Service

Wednesday, November 19 6:00 PM - 8:00 PM
StarTree Inc. Mountain View, California

About Event

Welcome to another edition of South Bay Systems! This time we bring you two wonderful talks: Julian Hyde will be speaking about Morel, a new functional database query language in development, and Yuanyuan Tian will be presenting the CIDR'25 paper on Query Optimization as a Service.

Agenda

6:00 PM: Doors open, food and socializing
6:30 PM — 6:50 PM: Morel Talk
6:50 PM — 7:30 PM: QOaaS Talk
7:30 PM onward : Community socializing!

Food and beverages will be provided, courtesy of our hosts, StarTree.

Morel: A language for data

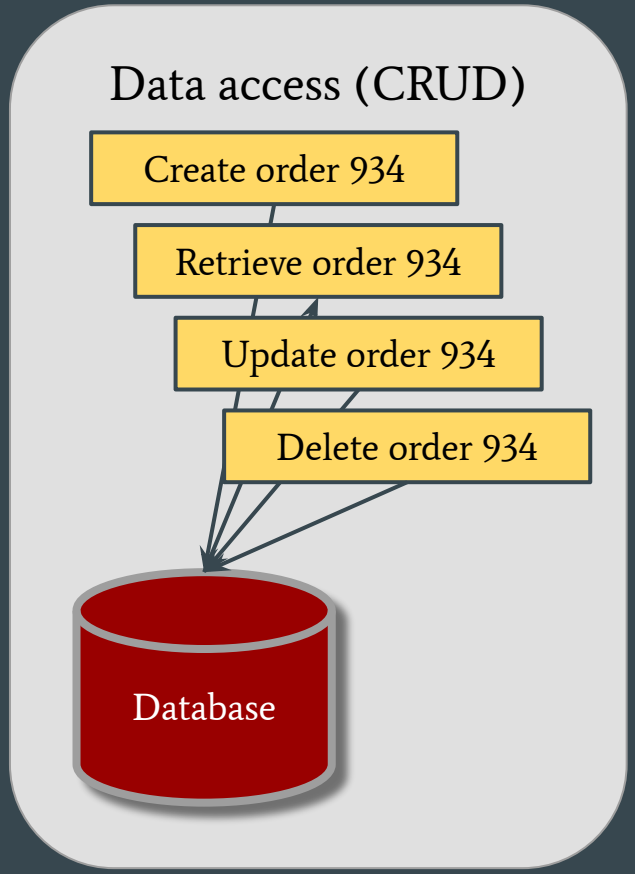
SQL excels at queries but struggles with streaming, incremental computation, version control, refactoring, and modern development workflows. Can we build a language that keeps SQL's strengths while addressing these limitations?

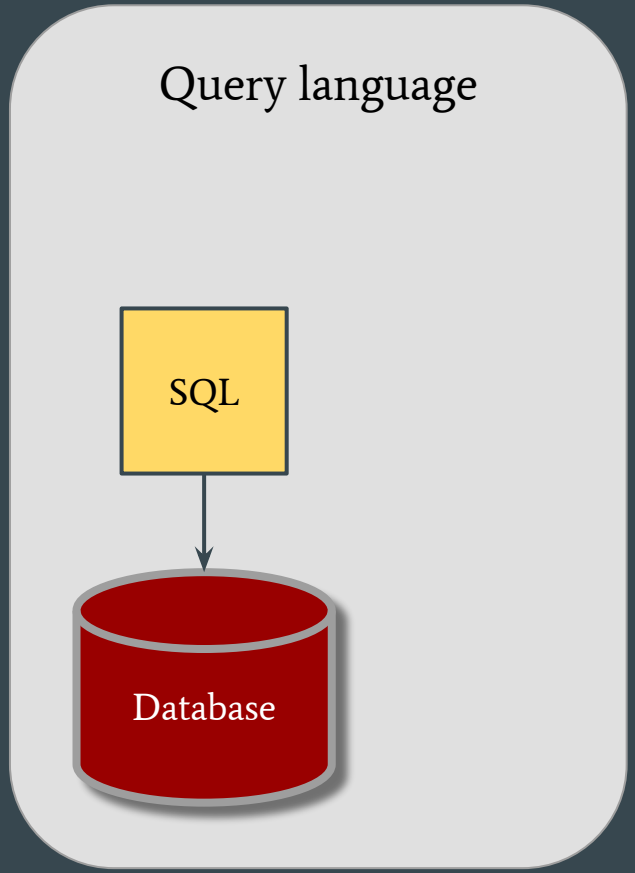
Morel combines functional programming with relational algebra to create a language as powerful as SQL but capable of solving a wider class of problems. This session introduces Morel and demonstrates how it addresses challenges like query federation, SQL dialect translation, streaming, and data engineering.

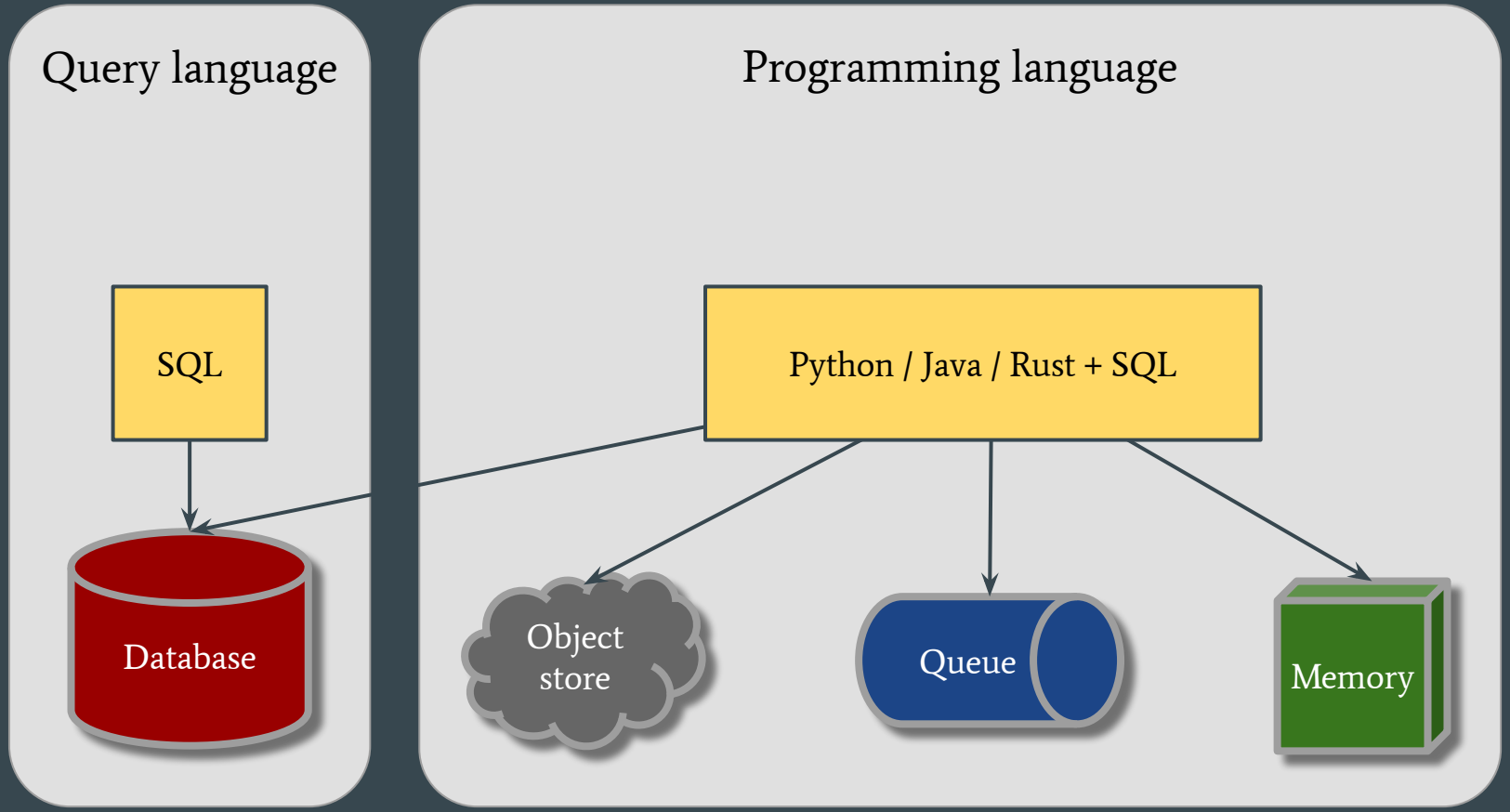
Speaker Bio

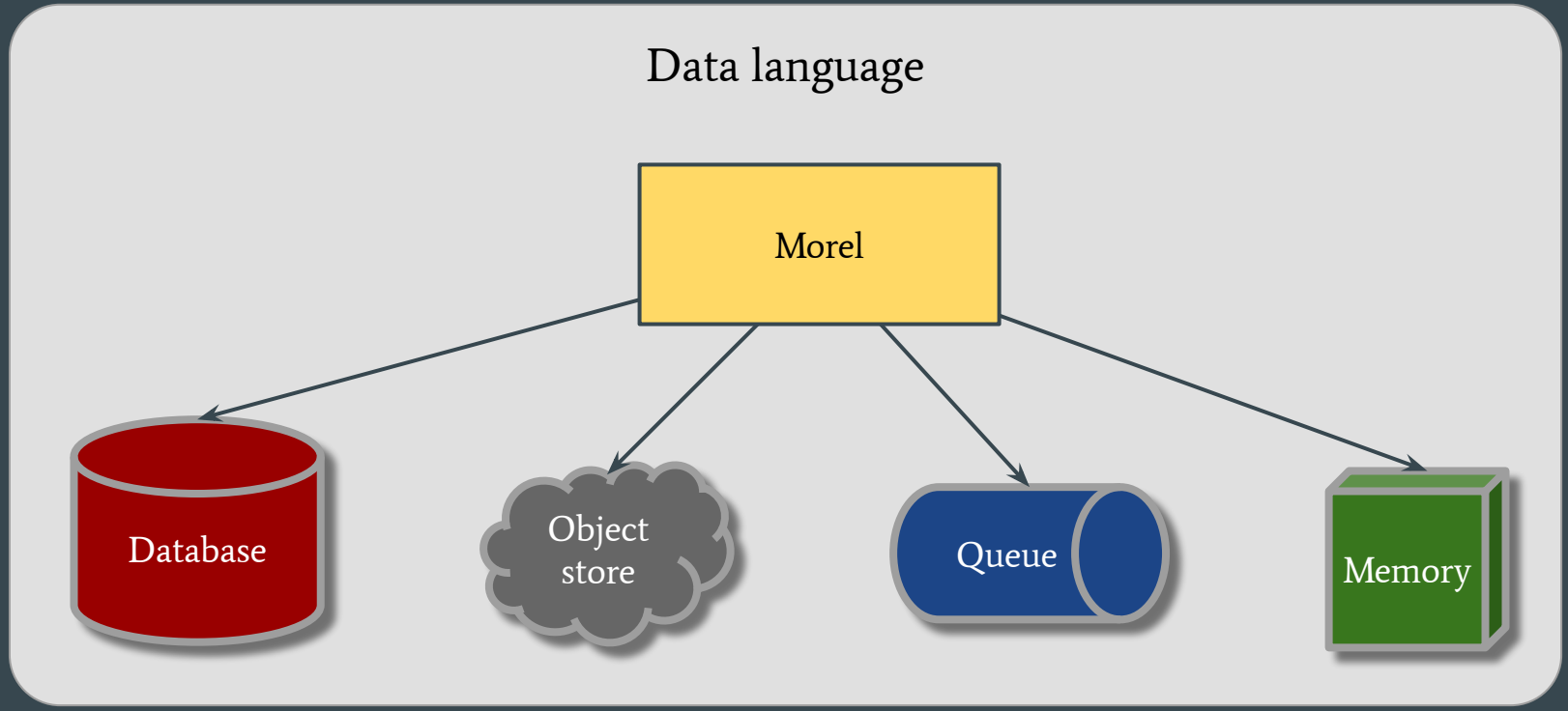
Julian Hyde is the author of Morel and creator of Apache Calcite, a widely-used open source query planning engine. He has pioneered SQL extensions for streaming and BI, and held senior engineering positions at Google and Hortonworks.

1. Databases & programming languages


Data access (CRUD)
Create order 934
Retrieve order 934
Update order 934
Delete order 934
Database


Query language
SQL
Database


Query language
Programming language
SQL
Python / Java / Rust + SQL
Database
Object store
Queue
Memory


Data language
Morel
Database
Object store
Queue
Memory

About me

Agenda

1. Databases & programming languages
2. Data language
3. Functional + relational
4. A high-level language
5. Best of both

2. Data language

What do we want from a data language?

1. Data

```
"Hello, world!";
> val it = "Hello, world!" : string

[1, 2, 3];
> val it = [1,2,3] : int list

(3.14, true);
> val it = (3.14, true) : real * bool

{empno = 100, ename = "SCOTT", job = "MANAGER"};
> val it : {empno:int,ename:string,job:string}

val depts = [
  {deptno = 10, dname = "SALES", emps = []},
  {deptno = 20, dname = "MARKETING", emps = [
    {empno = 100, ename = "SCOTT", job = "MANAGER"},
    {empno = 110, ename = "OATES", job = "CLERK"}
  ]}];
> val depts
  : {deptno:int,dname:string,
      emps:{empno:int, ename:string, job:string} list} list
```

What do we want from a data language?

1. Data
2. **Expressions**

```
substring ("abcde", 1, 2);
> val it = "bc" : string

tl [1, 2, 3];
> val it = [2,3] : int list

from i in [1, 2, 3, 4, 5]
 where i mod 2 = 1
 yield i * i;
> val it = [1,9,25] : int list

fun categorize (x, y) =
  case (x mod 2, y mod 2) of
      (0, 0) => "both even"
    | (1, 1) => "both odd"
    | (_, _) => "odd and even";
> val categorize = fn : int * int -> string
```

What do we want from a data language?

1. Data
2. Expressions
3. **Functions**

```
fn x => x * x;
> val it = fn : int -> int

map (fn x => x * x) [1, 2, 3, 4];
> val it = [1,4,9,16] : int list

fun factorial 1 = 1
  | factorial n = n * (factorial (n - 1));
> val factorial = fn : int -> int
```

What do we want from a data language?

1. Data
2. Expressions
3. Functions
4. **Types**

```
type employee =
  {empno:int, ename:string, is_mgr:bool,
   mgrno:int option};
> type employee

datatype color = BLUE | GREEN | RED;
> datatype color = BLUE | GREEN | RED

datatype 'a option = NONE | SOME of 'a;
> datatype option

SOME "abc";
> val it = SOME "abc" : string option

NONE;
> val it = NONE : 'a option
```

3. Functional + relational

Relational algebra in a functional programming language

Relational algebra

- ∪ union
- \ minus
- ∩ intersect
- σ filter
- Π project
- ⋈ join

Relational operators as functions

```
val union = fn : 'a list * 'a list -> 'a list
val except = fn : 'a list * 'a list -> 'a list
val intersect = fn : 'a list * 'a list -> 'a list
val filter = fn : ('a -> bool) -> 'a list -> 'a list
val map = fn : ('a -> 'b) -> 'a list -> 'b list
val join = fn
  : 'a list * 'b list * ('a * 'b -> bool)
    -> ('a * 'b) list
```

Chaining relational operators

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno}
  where nameLength > 4
  group deptno compute {c = count over (), s = sum over nameLength}
  where s > 10
  yield c + s;

> val it = [14] : int list
```

Chaining relational operators - step 1

```
from e in emps;
```

```
> val it =
  [{deptno=10,id=100,name="Fred"},
   {deptno=20,id=101,name="Velma"},
   {deptno=30,id=102,name="Shaggy"},
   {deptno=30,id=103,name="Scooby"}]
  : {deptno:int, id:int, name:string} list
```

Chaining relational operators - step 2

```
from e in emps
  order (e.deptno, DESC);
```

```
> val it =
  [{deptno=10,id=100,name="Fred"},
   {deptno=20,id=101,name="Velma"},
   {deptno=30,id=103,name="Scooby"},
   {deptno=30,id=102,name="Shaggy"}]
  : {deptno:int, id:int, name:string} list
```

Chaining relational operators - step 3

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno};
```

```
> val it =
  [{deptno=10,id=100,name="Fred",nameLength=4},
   {deptno=20,id=101,name="Velma",nameLength=5},
   {deptno=30,id=103,name="Scooby",nameLength=6},
   {deptno=30,id=102,name="Shaggy",nameLength=6}]
  : {deptno:int, id:int, name:string, nameLength:int} list
```

Chaining relational operators - step 4

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno}
  where nameLength > 4;



> val it =
  [{deptno=20,id=101,name="Velma",nameLength=5},
   {deptno=30,id=103,name="Scooby",nameLength=6},
   {deptno=30,id=102,name="Shaggy",nameLength=6}]
  : {deptno:int, id:int, name:string, nameLength:int} list
```

Chaining relational operators - step 5

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno}
  where nameLength > 4
  group deptno compute {c = count over (), s = sum over nameLength};



> val it =
  [{c=1,deptno=20,s=5},
   {c=2,deptno=30,s=12}]
  : {c:int, deptno:int, s:int} list
```

Chaining relational operators - step 6

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno}
  where nameLength > 4
  group deptno compute {c = count over (), s = sum over nameLength}
  where s > 10;


> val it =
    [{c=2,deptno=30,s=12}]
    : {c:int, deptno:int, s:int} list
```

Chaining relational operators

```
from e in emps
  order (e.deptno, DESC)
  yield {e.name, nameLength = size(e.name), e.id, e.deptno}
  where nameLength > 4
  group deptno compute {c = count over (), s = sum over nameLength}
  where s > 10
  yield c + s;

> val it = [14] : int list
```

Morel implementations

Toolchain

- Morel Java release 0.7
- Morel Rust release 0.2

Morel implementations

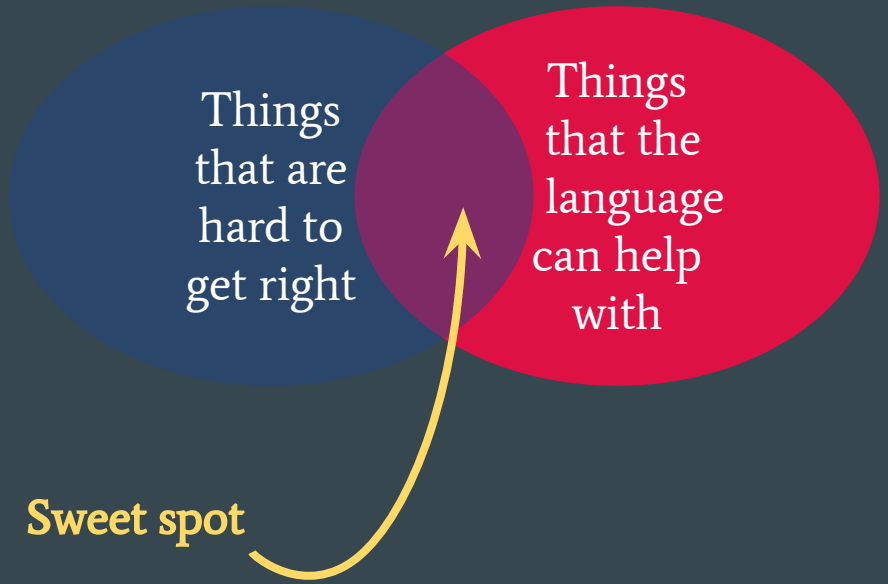
Toolchain

- Morel Java release 0.7
- Morel Rust release 0.2

Runtime

- Java interpreter & shell
- Rust interpreter & shell
- WebAssembly interpreter
- Various SQL dialects (via Apache Calcite)
- Apache Arrow/DataFusion (planned)

4. High-level language


Things that are hard to get right
Things that the language can help with
Sweet spot

What's a high-level language?

Part 1: Java ArrayList

```
record LogEntry(String timestamp, String message) {}

List<LogEntry> logsList = List.of(
    new LogEntry("2025-10-25T08:15:23", "User login: alice"),
    new LogEntry("2025-10-25T08:16:45", "API call: /users"),
    new LogEntry("2025-10-25T09:23:11", "Error: timeout"),
    new LogEntry("2025-10-25T10:05:33", "User login: bob"),
    new LogEntry("2025-10-25T14:22:01", "API call: /orders")
    // ... millions of log entries
    );

List<String> logsInRange(String startTime,
   String endTime, List<LogEntry> logs) {
  var result = new ArrayList<String>();
  for (var entry : logs) {
    if (entry.timestamp.compareTo(startTime) >= 0
        && entry.timestamp.compareTo(endTime) <= 0) {
      result.add(entry.message);
    }
  }
  return result;
}

logsInRange("2025-10-2509:00:00", "2025-10-2511:00:00", logsList);
```

What's a high-level language?

Part 2: Java SortedMap

```
SortedMap<String, String> logsMap =
    new TreeMap<>(
        Map.ofEntries(
            Map.entry("2025-10-25T08:15:23", "User login: alice"),
            Map.entry("2025-10-25T08:16:45", "API call: /users"),
            Map.entry("2025-10-25T09:23:11", "Error: timeout"),
            Map.entry("2025-10-25T10:05:33", "User login: bob"),
            Map.entry("2025-10-25T14:22:01", "API call: /orders")
            // ...
        ));

List<String> logsInRange(String startTime,
    String endTime, SortedMap<String, String> logs) {
  var result = new ArrayList<String>();
  var subMap = logs.subMap(startTime, true, endTime, true);
  for (var message : subMap.values()) {
    result.add(message);
  }
  return result;
}

logsInRange("2025-10-2509:00:00", "2025-10-2511:00:00", logsMap);
```

What's a high-level language?

Part 3: Rust BTreeMap

```
let mut logs_map = BTreeMap::new();
logs_map.insert("2025-10-25T08:15:23".to_string(),
    "User login: alice".to_string());
logs_map.insert("2025-10-25T08:16:45".to_string(),
    "API call: /users".to_string());
logs_map.insert("2025-10-25T09:23:11".to_string(),
    "Error: timeout".to_string());
logs_map.insert("2025-10-25T10:05:33".to_string(),
    "User login: bob".to_string());
logs_map.insert("2025-10-25T14:22:01".to_string(),
    "API call: /orders".to_string());

fn logs_in_range(start_time: &str, end_time: &str,
    logs: &BTreeMap<String, String>) -> Vec<String> {
    let mut result = Vec::new();
    for (_timestamp, message)
      in logs.range(start_time.to_string()..=end_time.to_string()) {
        result.push(message.clone());
    }
  result
}

logs_in_range("2025-10-2509:00:00", "2025-10-2511:00:00", logsList);
```

What's a high-level language?

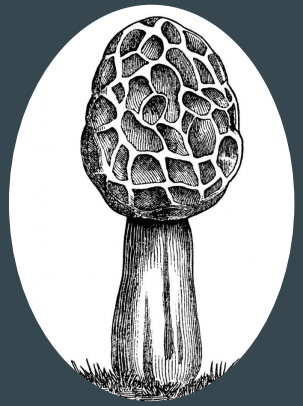
Part 4: Morel list

```
val logsList = [
  ("2025-10-25T08:15:23", "User login: alice"),
  ("2025-10-25T08:16:45", "API call: /users"),
  ("2025-10-25T09:23:11", "Error: timeout"),
  ("2025-10-25T10:05:33", "User login: bob"),
  ("2025-10-25T14:22:01", "API call: /orders")
  (* ... millions of log entries *)
];

fun logsInRange (startTime, endTime, logs) =
  from (timestamp, message) in logs
    where timestamp >= startTime
      andalso timestamp <= endTime
    yield message;

logsInRange ("2025-10-2509:00:00", "2025-10-2511:00:00", logsList);
```

high-level programming language *n*. A programming language that requires you to specify only the details that matter.

5. Best of both

Relational languages



Functional programming languages

Best of functional programming languages

General-purpose

If it compiles, it probably works

Refactoring & autocompletion

Git

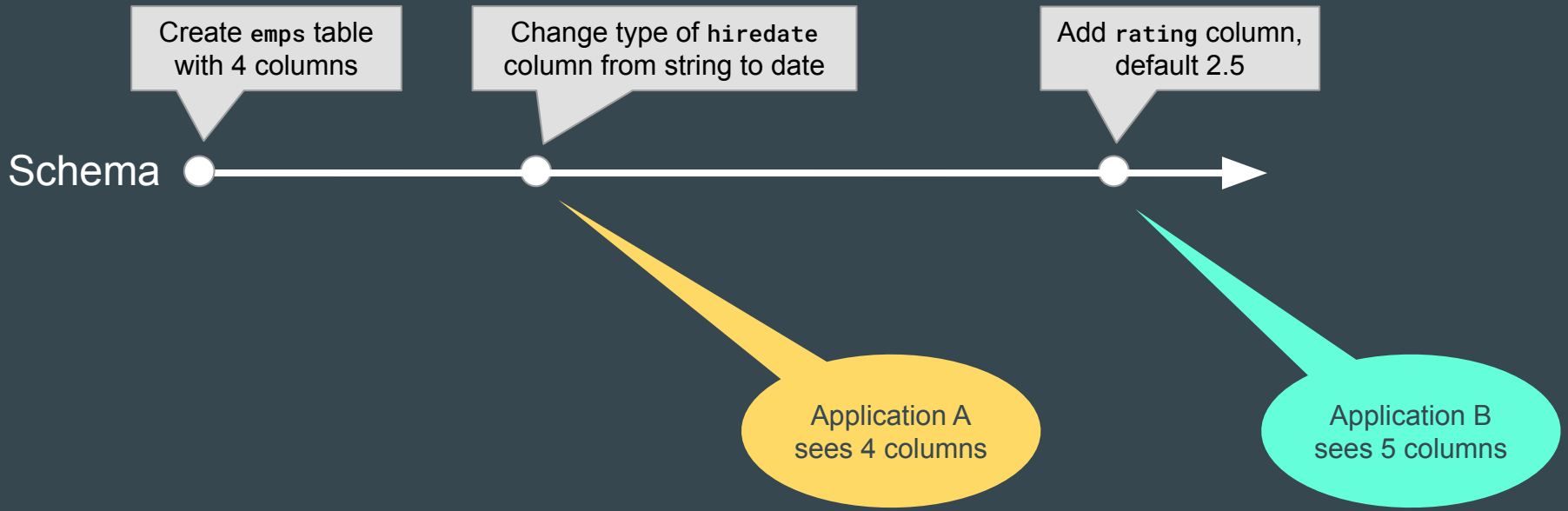
Documentation in the code

Unit tests in the same language

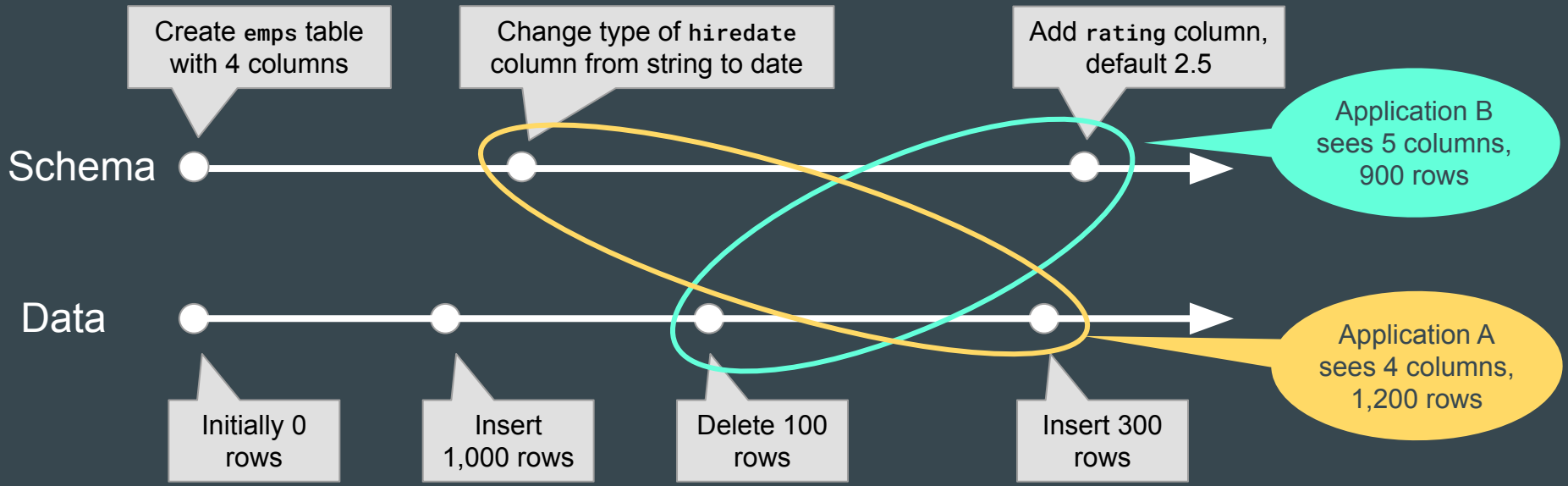
Modules & versioning

Abstraction

Schema evolution



Types and data evolve independently



Best of relational

Parallel/distributed execution

Algebraic optimization

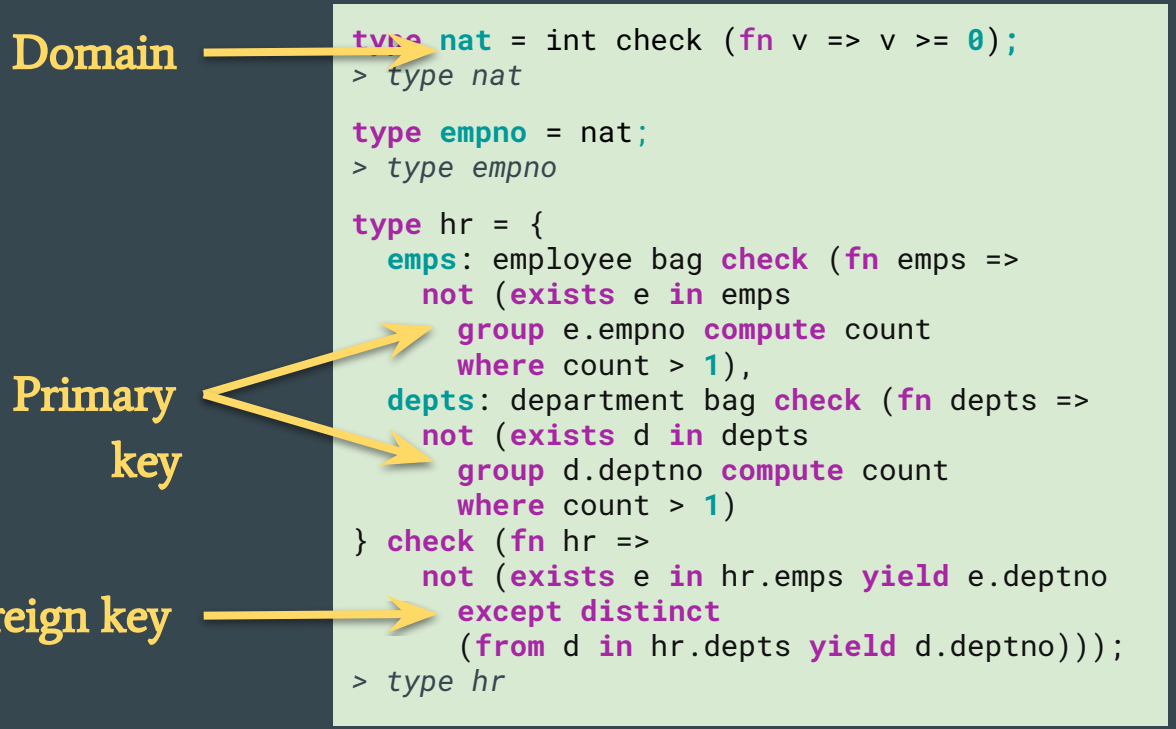
Optimize data structures

Incremental computation

Constraints

Derived data

Views



Constraints, derived data, and views

Constraint

```
check (from e in products unorder)
   = (from e in products_by_mfr unorder)
```

Derived data

```
CREATE MATERIALIZED VIEW products_by_mfr AS
  SELECT *
  FROM products
  ORDER BY mfr
```

View

```
fun products_by_mfr () =
  from p in products
  order p.mfr
```

Differences

- What happens when I try to insert a row into one data set but not the other?
- Can I define a lossy view? (e.g. orders group by week, zip code)
- Can I define a denormalized view? (e.g. orders with nested order-items)

Best of both

Relational

Efficient parallel/distributed execution

Algebraic optimization

Optimize data structures

Incremental computation

Constraints

Derived data

Views

Functional

General-purpose

If it compiles, it probably works

Refactoring & autocompletion

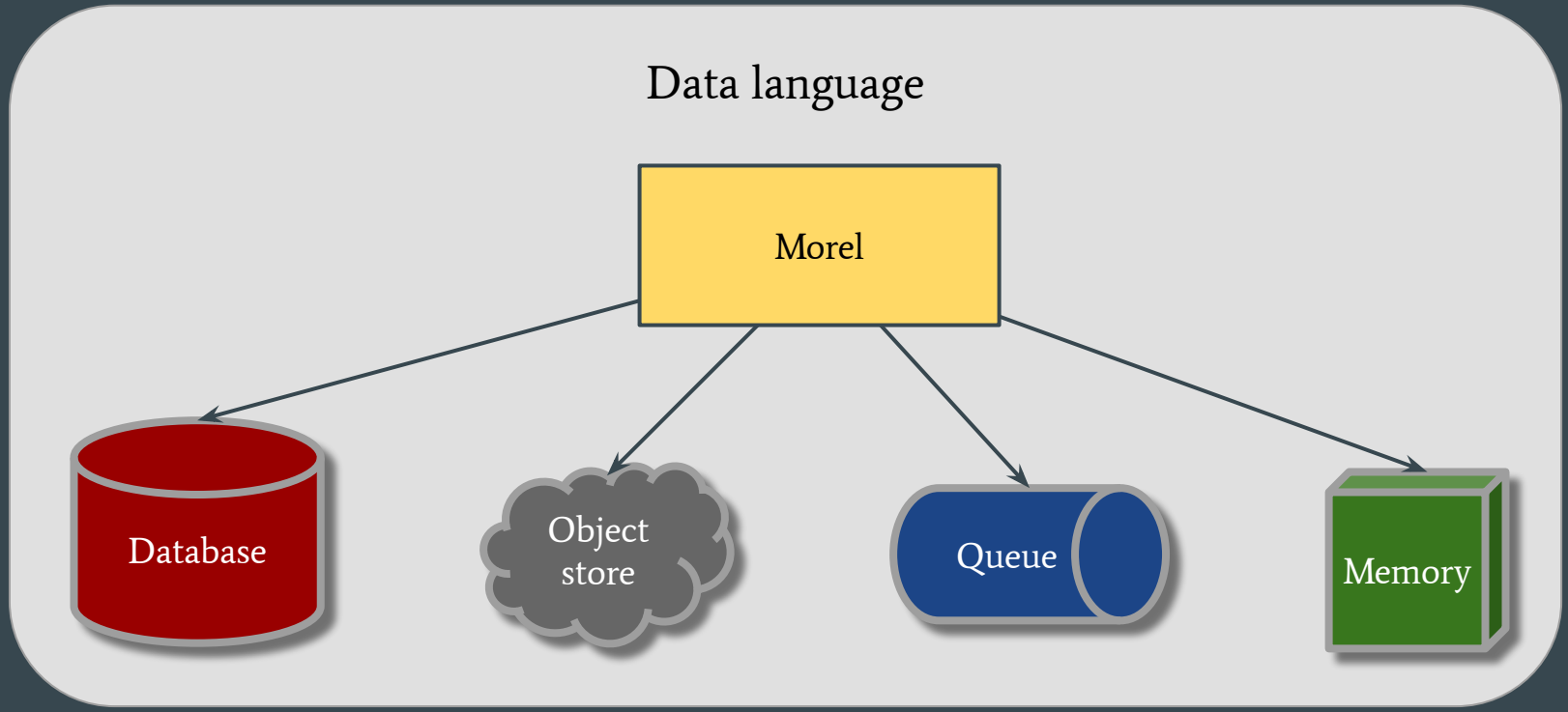
Git

Documentation in the code

Unit tests in the same language

Modules & versioning

Abstraction


Data language
Morel
Database
Object store
Queue
Memory

Morel: A language for data

@julianhyde · @morel_lang
https://github.com/julianhyde · https://github.com/hydromatic/morel · https://github.com/hydromatic/morel-rust