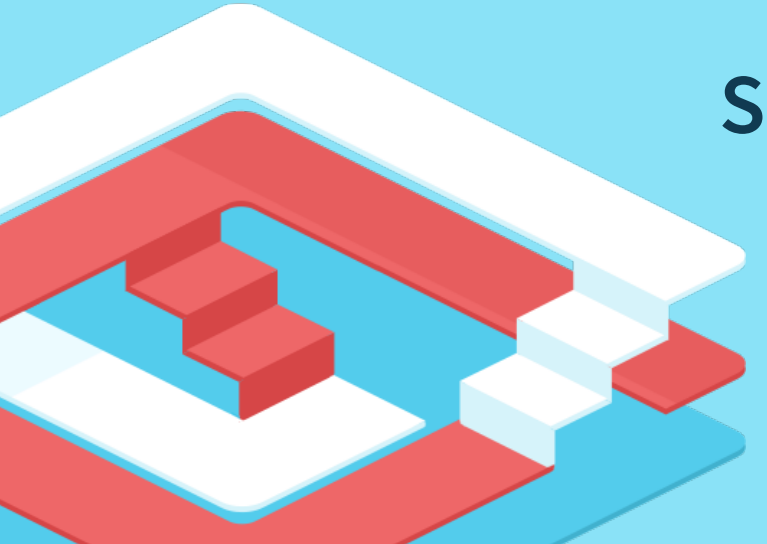
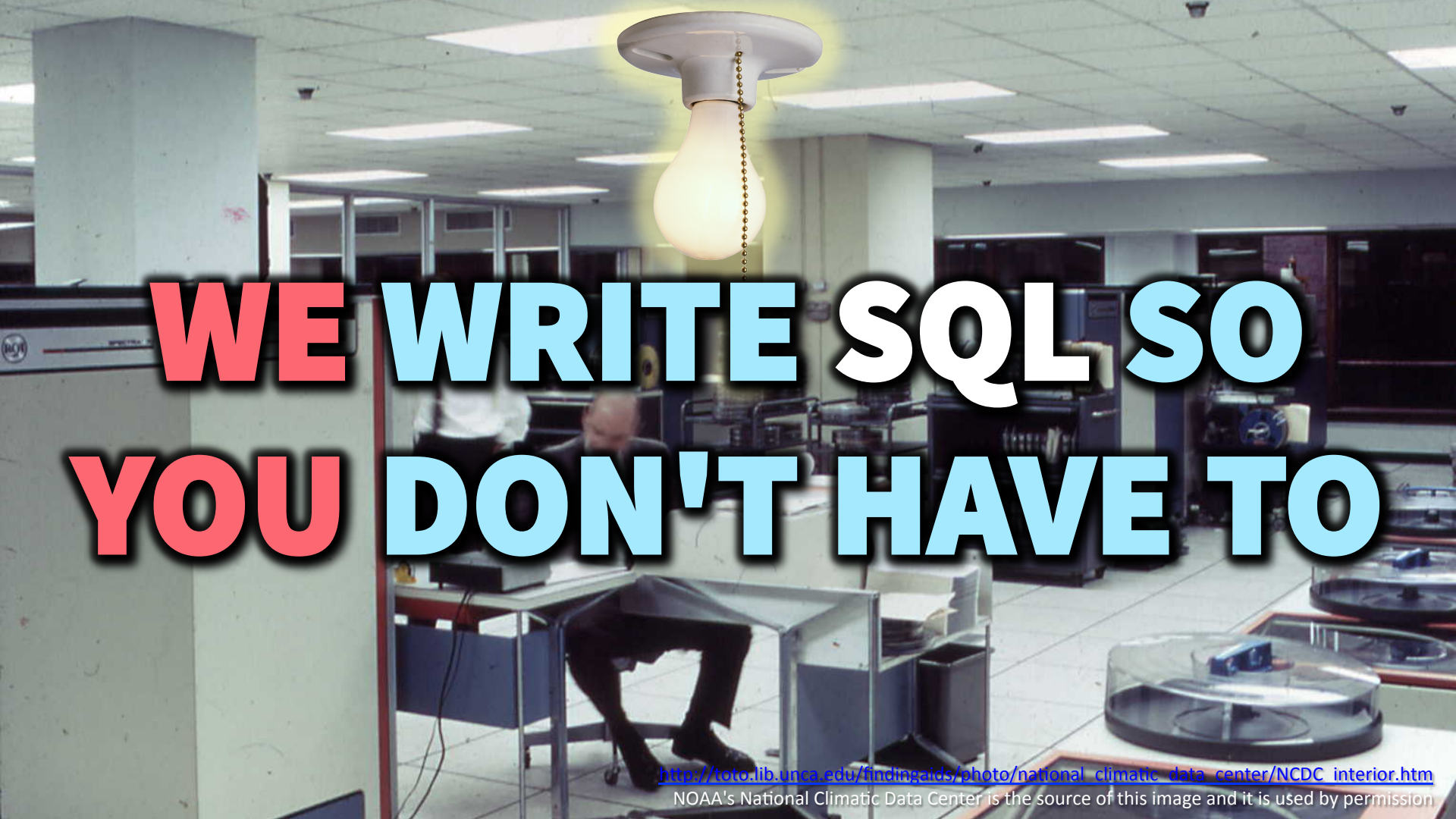
Scaling Scala to the Database

Stefan Zeiger



Typesafe

1
Overview / Key Concepts



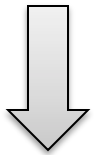
http://toto.lib.unca.edu/findingaids/photo/national_climatic_data_center/NCDC_interior.htm

NOAA's National Climatic Data Center is the source of this image and it is used by permission

Write database code in Scala

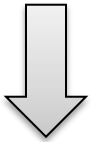
- Instead of SQL, JPQL, Criteria API, etc.

```
for { p <- persons } yield p.name
```



```
select p.NAME from PERSON p
```

```
(for {
      p <- persons.filter(_.age < 20) ++
           persons.filter(_.age >= 50)
           if p.name.startsWith("A")
} yield p).groupBy(_.age).map { case (age, ps) =>
  (age, ps.length)
}
```



```
select x2.x3, count(1) from (
  select * from (
    select x4."NAME" as x5, x4."AGE" as x3
      from "PERSON" x4 where x4."AGE" < 20
    union all select x6."NAME" as x5, x6."AGE" as x3
      from "PERSON" x6 where x6."AGE" >= 50
    ) x7 where x7.x5 like 'A%' escape '^'
  ) x2 group by x2.x3
```

Scala Language Integrated Connection Kit

- Database query and access library for Scala
- Successor of ScalaQuery
- Developed at Typesafe and EPFL
- Open Source

Supported Databases

- PostgreSQL
- MySQL
- H2
- Hsqldb
- Derby / JavaDB
- SQLite
- Access

Closed-Source *Slick Extensions* (with commercial support by Typesafe):

- Oracle
- DB/2
- SQL Server

Components

- Lifted Embedding
- Direct Embedding
- Plain SQL
- Session Management
- Schema Model

2
Compared to ORMs

Impedance Mismatch: Retrieval


```
select COF_NAME
from COFFEES
```

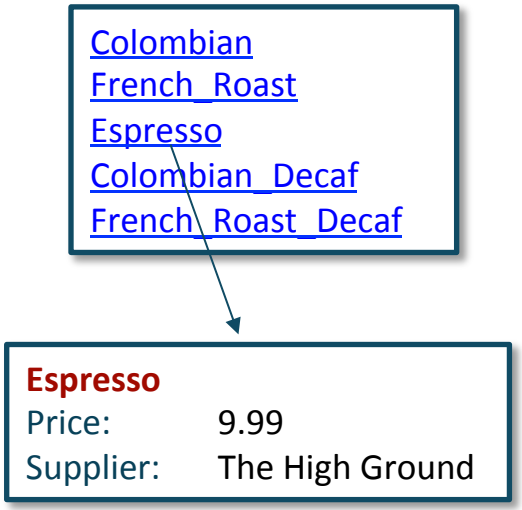
```
select c.*, s.SUP_NAME
from COFFEES c, SUPPLIERS s
where c.COF_NAME = ?
and c.SUP_ID = s.SUP_ID
```

Impedance Mismatch: Retrieval

```
def getAllCoffees(): Seq[Coffee] = …

def printLinks(s: Seq[Coffee]) {
  for(c <- s) println(c.name + " " + c.price )
}

def printDetails(c: Coffee) {
  println(c.name)
  println("Price: " + c.price)
  println("Supplier: " + c.supplier.name)
}
```



O/R Mapper

- Mapping low-level programming (OOP) to high-level concepts (relational algebra)

- Not transparent (but pretends to be)

Better Match: Functional Programming

- Relation
- Attribute
- Tuple
- Relation Value
- Relation Variable - mutable state in the DB

```
case class Coffee(name: String,
  supplierId: Int, price: Double)

val coffees = Set(
    Coffee("Colombian",    101, 7.99),
    Coffee("French_Roast", 49, 8.99),
    Coffee("Espresso",     150, 9.99)
  )
```

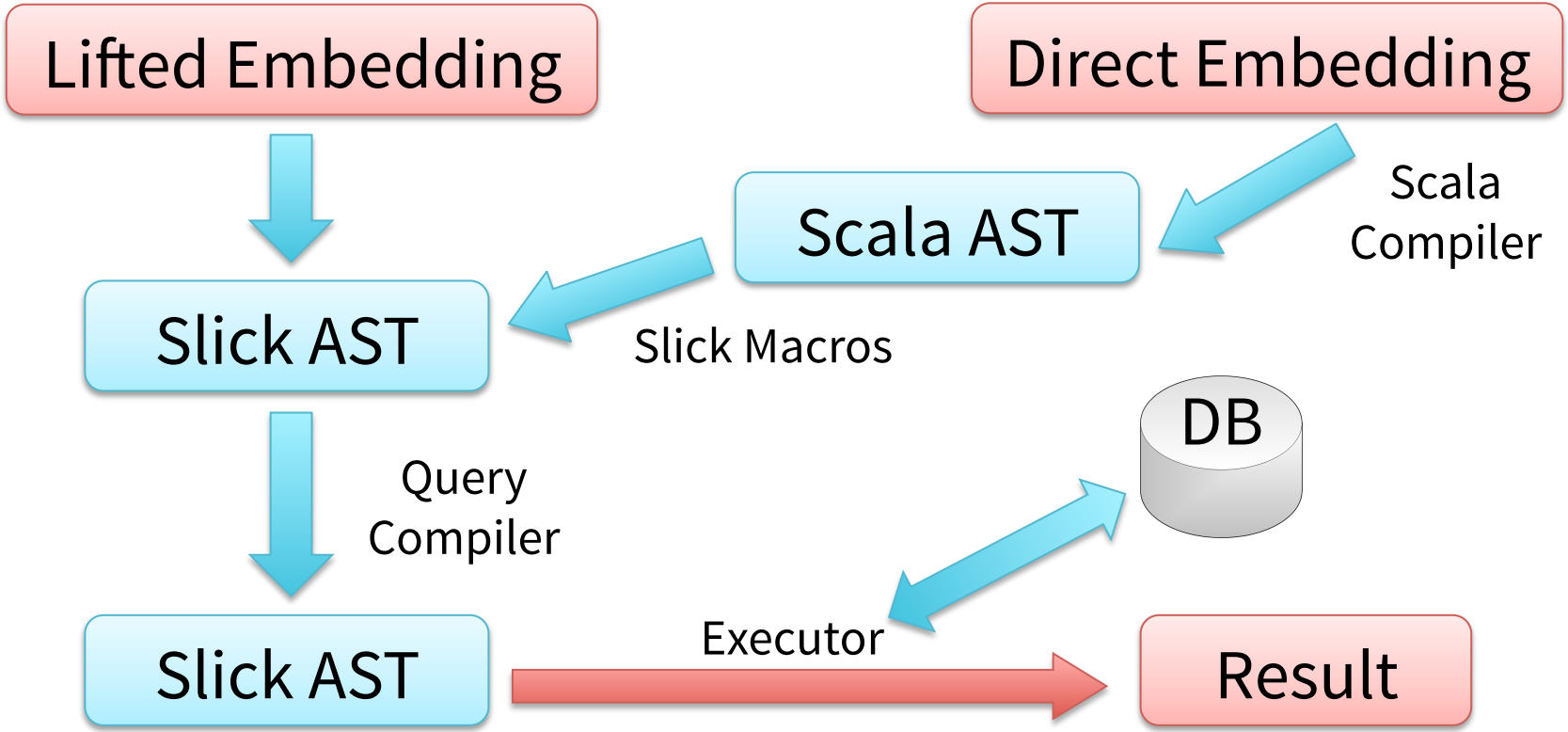
Functional-Relational Mapping

- Embraces the relational model
- No impedance mismatch
- Composable Queries
- Explicit control over statement execution
- Stateless

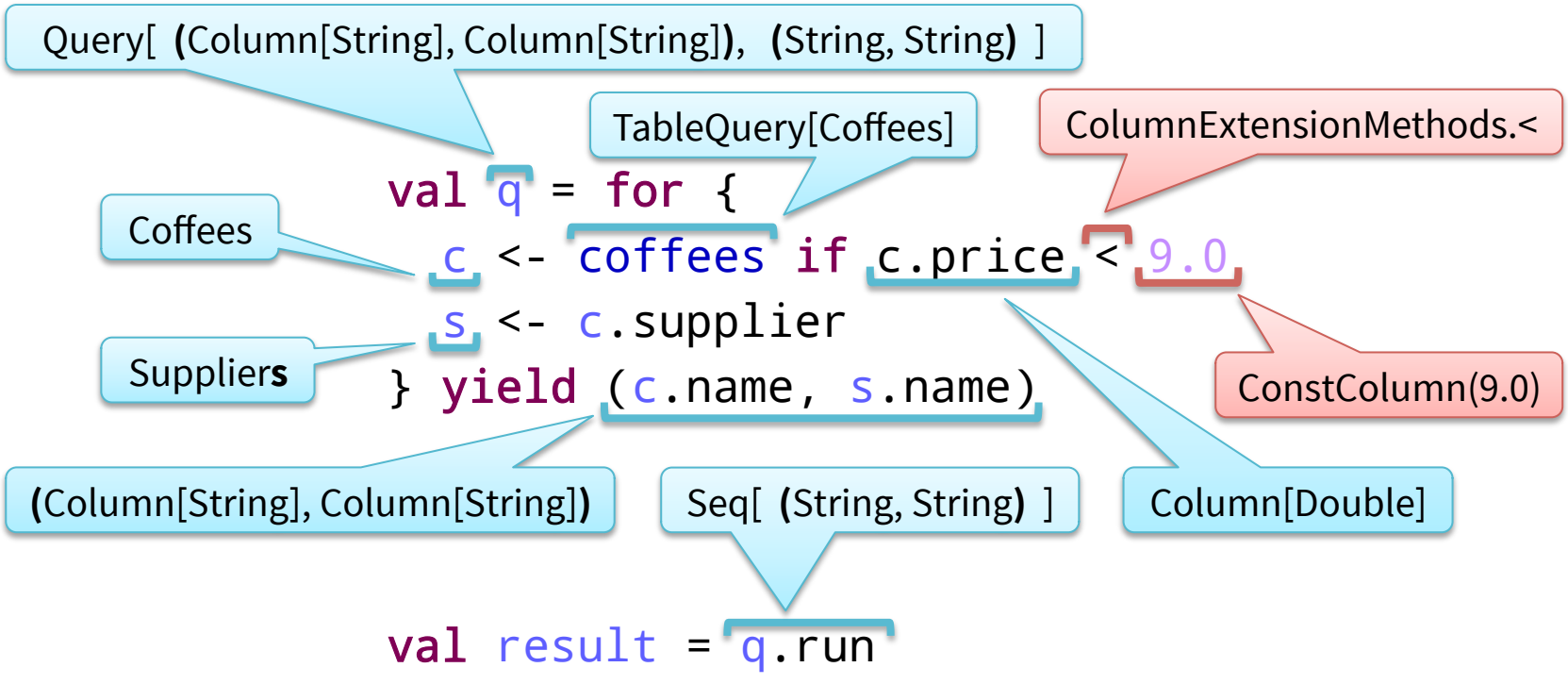
3
Demo

4
Under The Hood

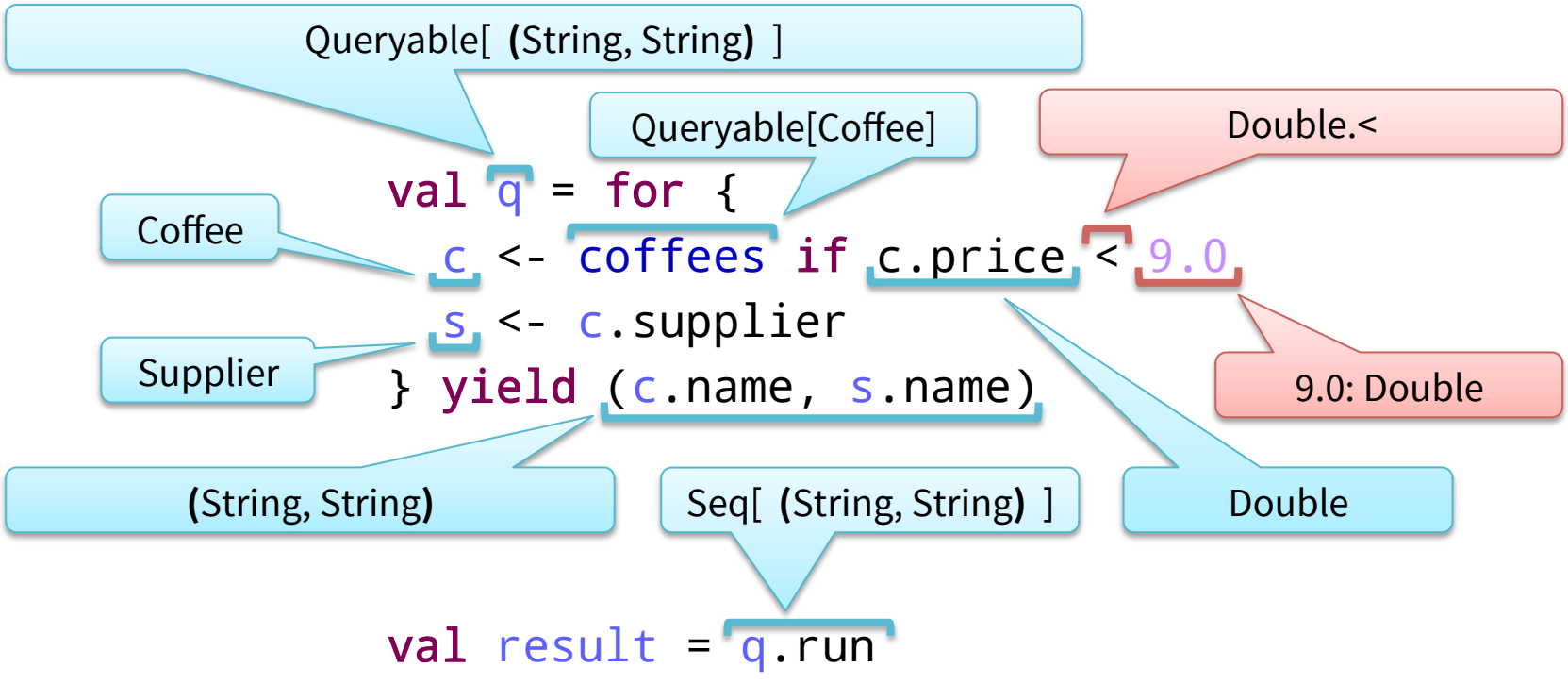
APIs


Lifted Embedding
Direct Embedding
Scala AST
Scala Compiler
Slick AST
Slick Macros
DB
Query Compiler
Slick AST
Executor
Result

Lifted Embedding



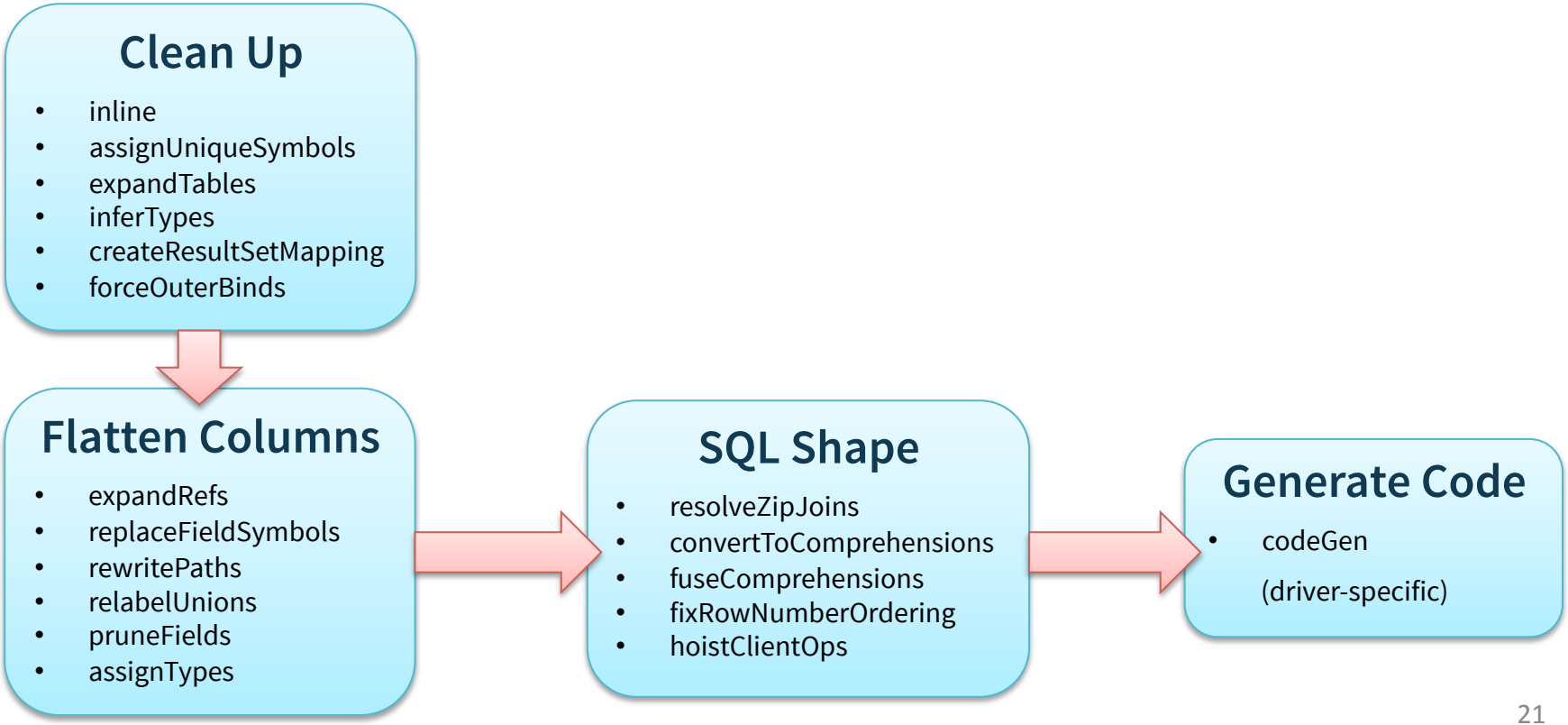
Direct Embedding (experimental)



Query Compiler

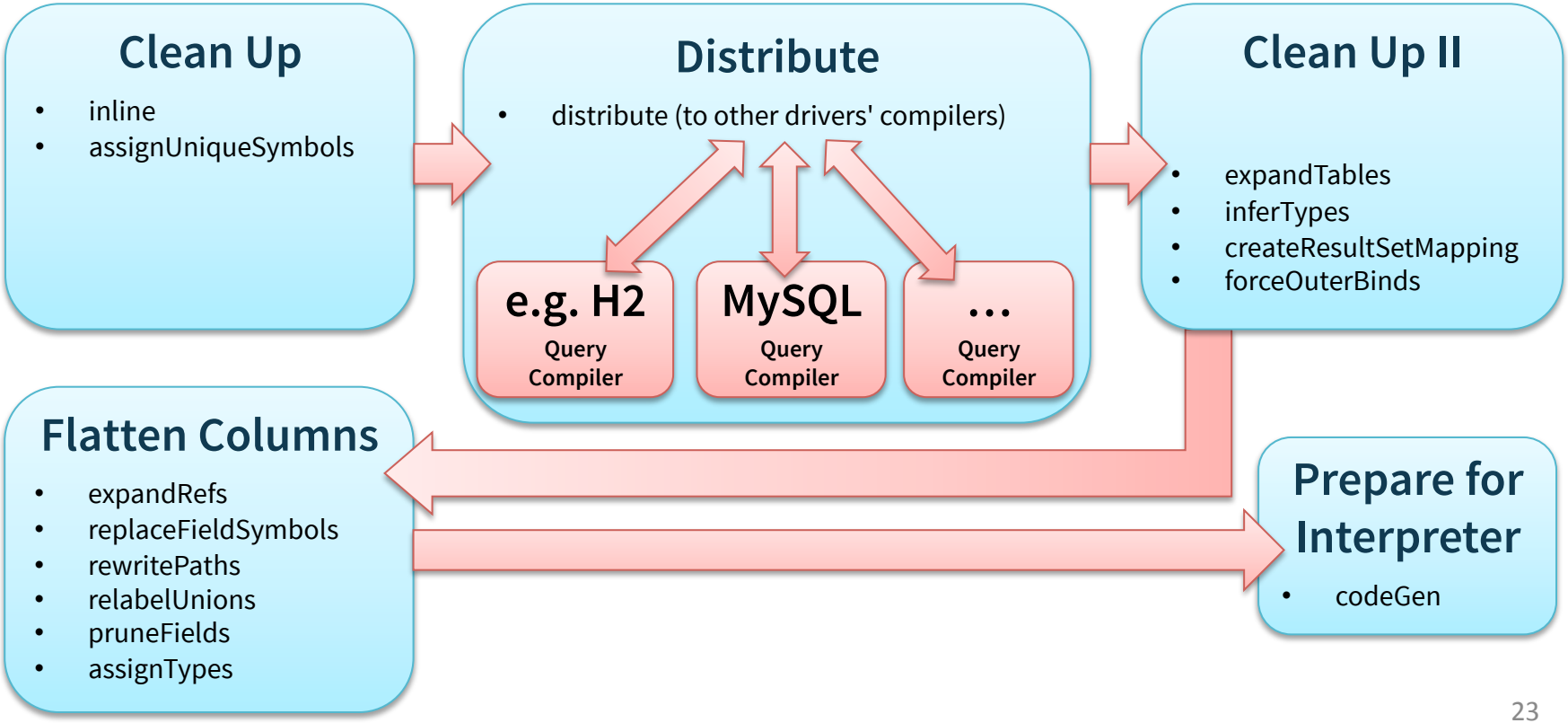
- Immutable ASTs
  - Types can be mutated until they are observed
- Immutable compiler state
  - containing AST + phase output state
- Phases transform compiler state
  - using mutable state locally
- Drivers provide their own compilers

Compiler Phases: SQL



Compiler Phases: MemoryDriver

Compiler Phases: Scheduling



5
Outlook

Slick 2.0

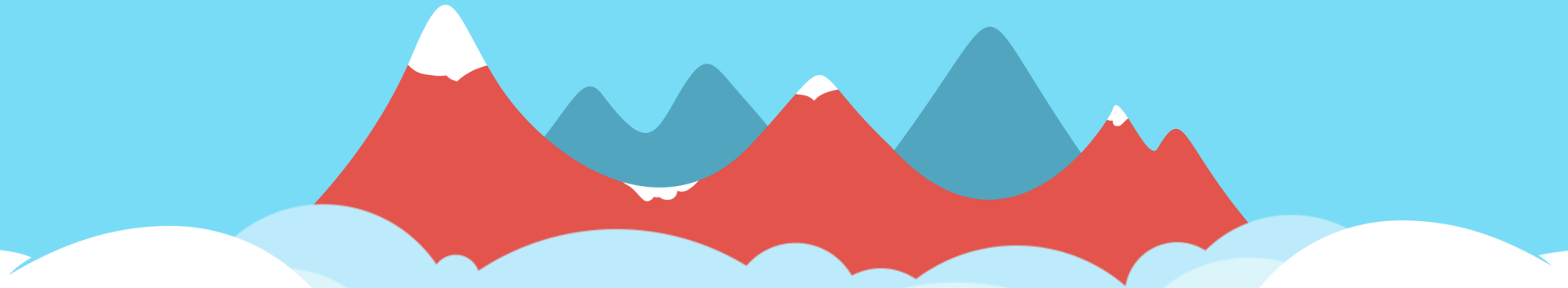
- Coming Q4 / 2013
- Query scheduling
- API Improvements
- New driver and back-end architecture
- Generate Slick code from database schemas

Outlook

- Macro-based type providers
  - Prototype based on type macros (*topic/type-providers*)
  - Released version will use macro annotations
  - Scala 2.12?
- Default database library for Play
  - as part of the Typesafe Reactive Platform
- Focus on usability (API, docs, semantics, etc.)

slick.typesafe.com

@StefanZeiger



Typesafe