

Slick database access with Scala

Stefan Zeiger



Your App And Your Database



[Image by Don & Tonya Christner](#)

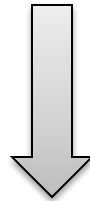


[Image by Lxowle](#)

Idea

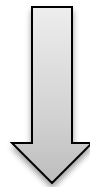
- **Write your database code in Scala**
 - Instead of SQL, JPQL, Criteria API, etc.

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



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

```
(for {  
  p <- Persons.filter(_.age < 20) unionAll  
    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
```

Agenda

- **Key Concepts**
- Live Demo
- Under The Hood
- Outlook

Slick

Scala Language Integrated Connection Kit

- Database query and access library for Scala
- Successor of ScalaQuery
- Developed at Typesafe and EPFL
- Version 0.11 launched in August
- 1.0 to be released shortly after Scala 2.10
- Use ScalaQuery 0.11-M1 for Scala 2.9 instead

Supported Databases

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

Closed-Source Slick Extensions
(commercially supported by
Typesafe) to be released with 1.0:

- Oracle
- DB/2

Next big step: NoSQL!
MongoDB support coming
Q1/2013

Why not use an ORM tool?

A photograph of several soldiers in a field, some sitting and some standing, with military equipment scattered on the ground. The soldiers are wearing green uniforms and helmets. One soldier in the foreground is sitting on the ground, looking towards the camera. Another soldier is sitting next to him, and a third is standing to the right, looking towards the left. There are various pieces of military equipment, including a helmet and a bag, scattered on the ground. The background shows a grassy field with some trees and hills in the distance.

“Object/Relational Mapping is The Vietnam of Computer Science”

(Ted Neward)

<http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>

Impedance Mismatch: Concepts

Object-Oriented:

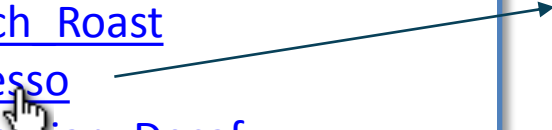
- Identity
- State
- Behaviour
- Encapsulation

Relational:

- Identity
- State: Transactional
- Behaviour
- Encapsulation

Impedance Mismatch: Retrieval

[Colombian](#)
[French Roast](#)
[Espresso](#)
[Colombian Decaf](#)
[French Roast Decaf](#)

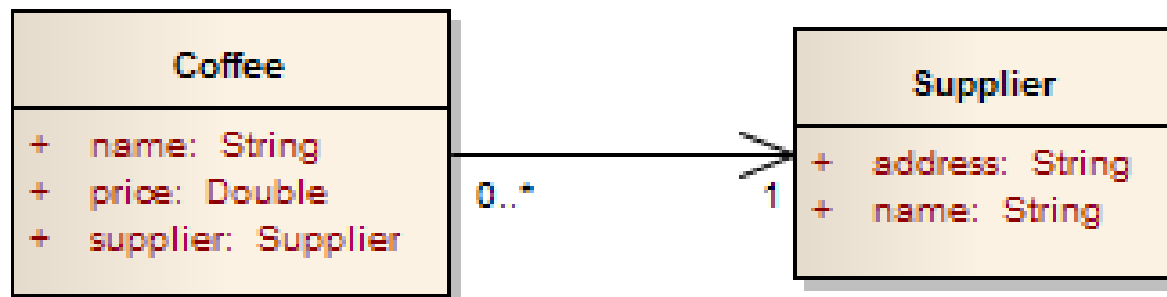
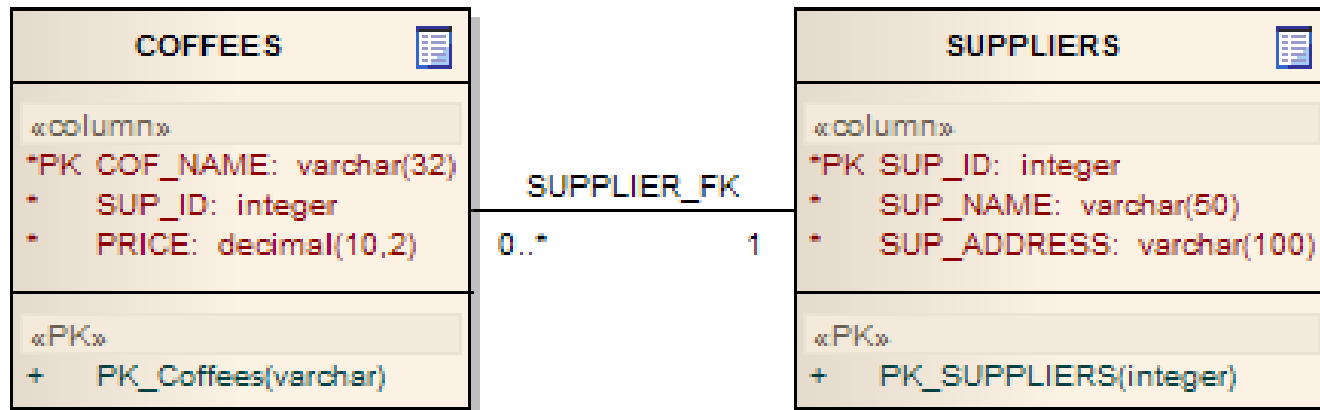


Espresso
Price: 9.99
Supplier: The High Ground

```
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



Impedance Mismatch: Retrieval



```
def getAllCoffee(): Seq[Coffee] = ...

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

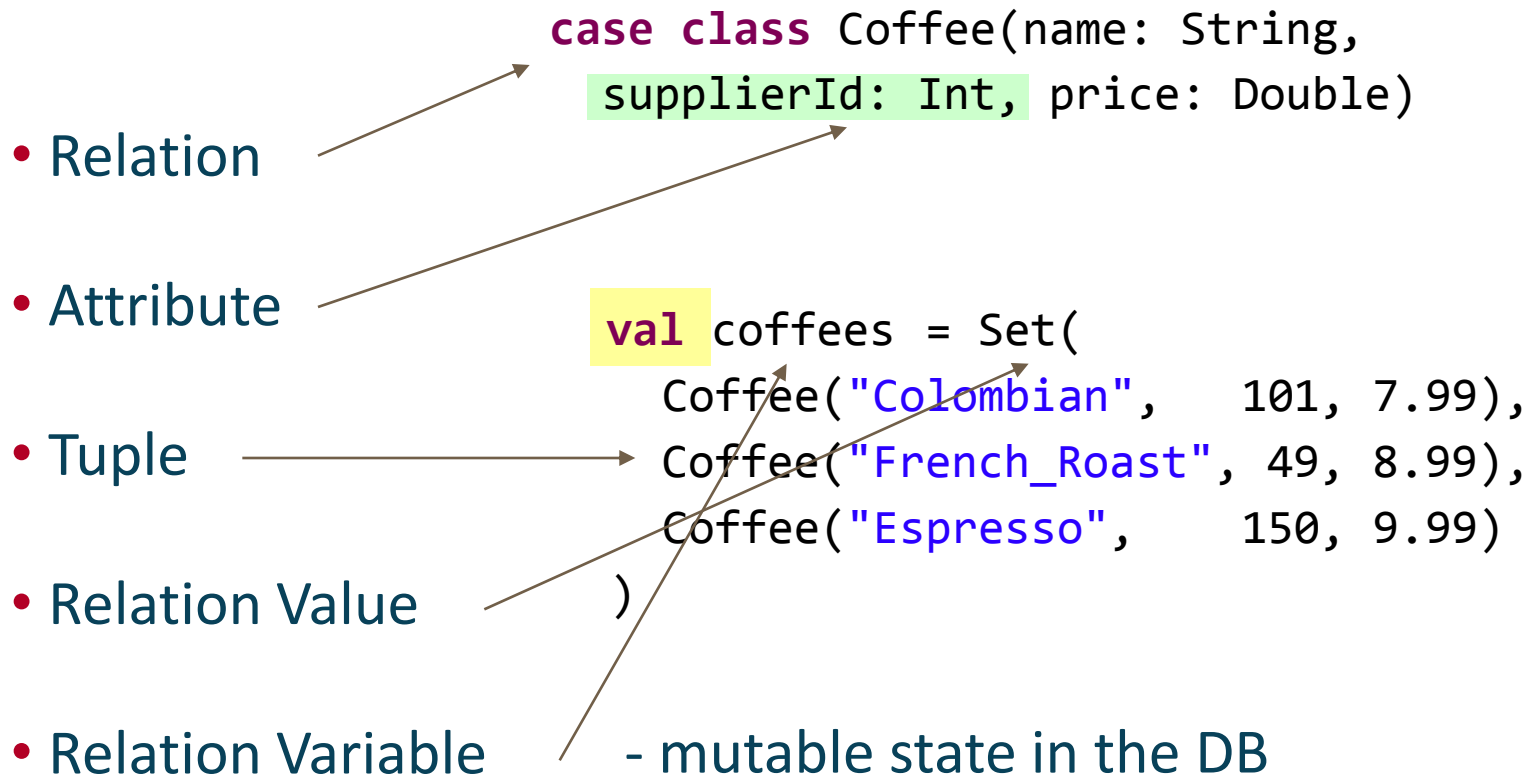
def printDetails(c: Coffee) {
  println(c.name)
  println("Price: " + c.price)
  println("Supplier: " + c.supplier.name)
}
```

O/R Mapper

- Mapping low-level programming (OOP) to high-level concepts (relational algebra)
- Not transparent

Better Match: Functional Programming

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



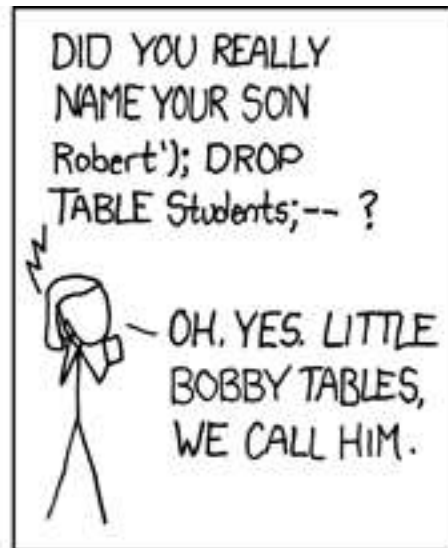
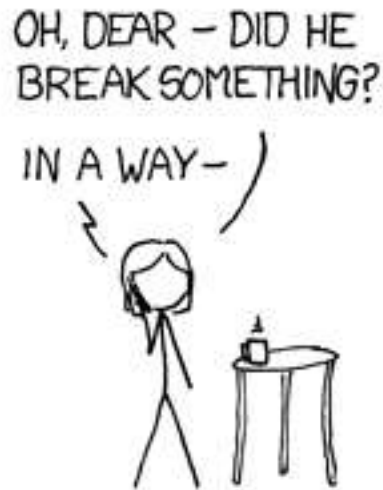
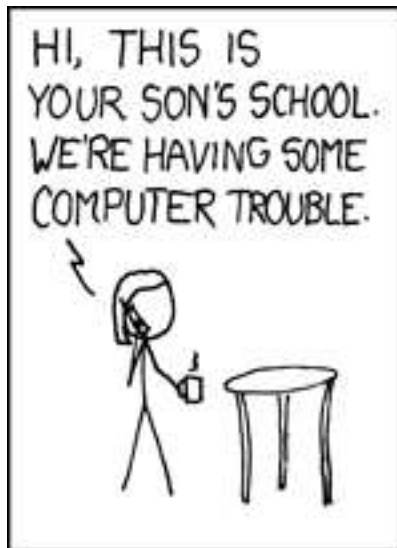
# Compared to ORMs

- **Slick is simple!**
  - Just write your queries in Scala
- **Slick is explicit!**
  - No lazy loading means predictable performance
  - Only read the data you need
- **Slick is functional!**
  - No mutable state (except in the database)

Why not write your own SQL code?

# SQL

- **Non-compositional** syntax
- Generating SQL via string manipulation is awkward
- Generating it from templates (e.g. MyBatis) is verbose
- Easy to make mistakes which are not caught at compile-time



# Compared to SQL

- **Slick is simple!**
  - Just write your queries in Scala
- **Slick is compositional!**
  - Not based on ad-hoc syntax and semantics
- **Slick is safe!**
  - Protects against type errors, spelling mistakes, wrong composition, etc.

# Plain SQL Queries



```
def personsMatching(pattern: String)(conn: Connection) = {
 val st = conn.prepareStatement(
 "select id, name from person where name like ?")
 try {
 st.setString(1, pattern)
 val rs = st.executeQuery()
 try {
 val b = new ListBuffer[(Int, String)]
 while(rs.next)
 b.append((rs.getInt(1), rs.getString(2)))
 b.toList
 } finally rs.close()
 } finally st.close()
}
```

# Plain SQL Queries



```
def personsMatching(pattern: String)(implicit session: Session) =
 sql"select id, name from person where name like $pattern"
 .as[(Int, String)].list
```

# Agenda

- Key Concepts
- **Live Demo**
- Under The Hood
- Outlook



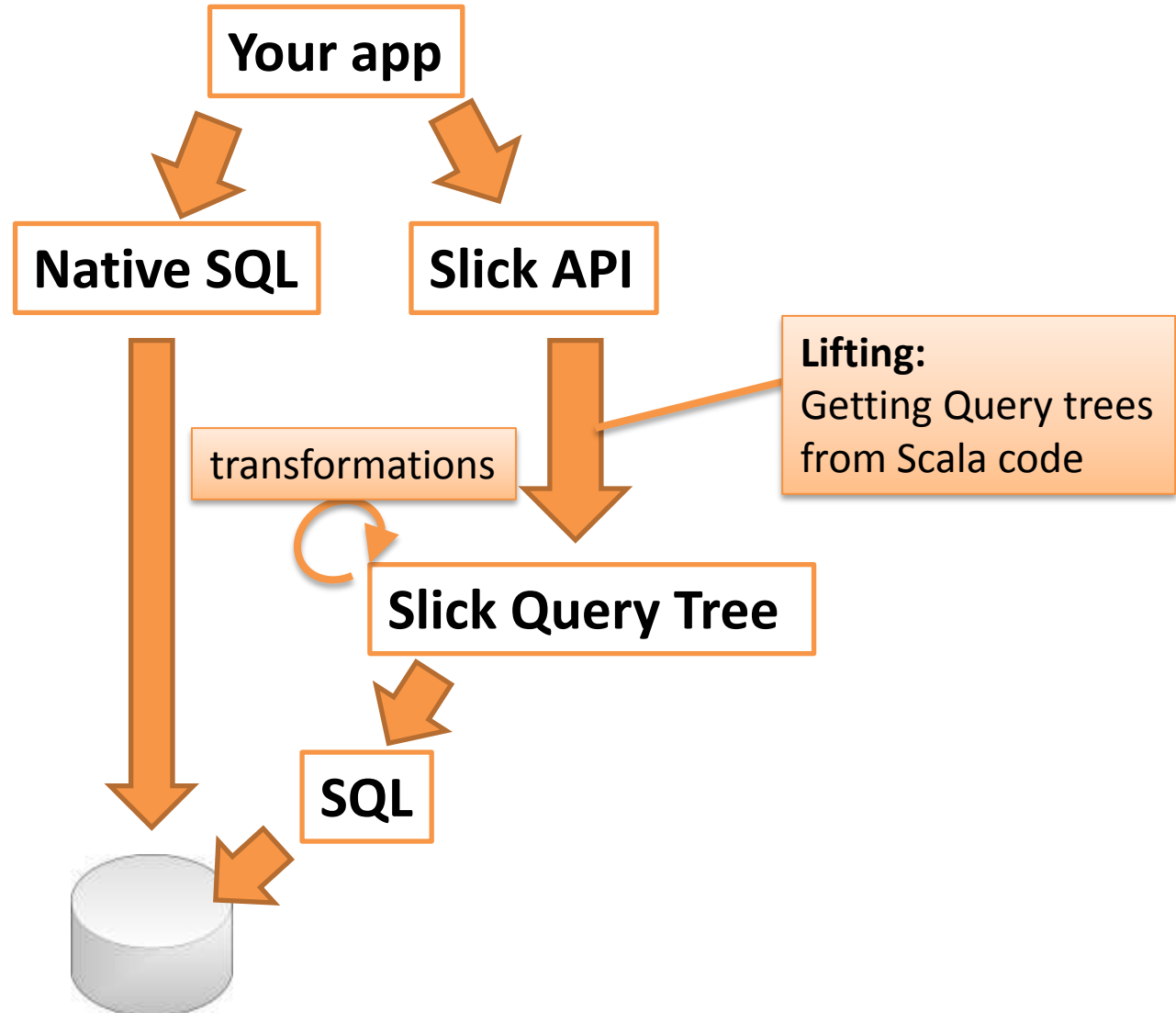
# Live Demo

- Clone it from <https://github.com/szeiger/slick-scalaexchange2012>
- Scaffolding, tables, mapping, insert
- Query, map, getting results, printing statements
- Comprehension, implicit join, sortBy, table methods, foreign keys
- Finders, foreach, bind variables, templates
- Implicit join, pagination, outer join, Option
- groupBy

# Agenda

- Key Concepts
- Live Demo
- **Under The Hood**
- Outlook

# Under the hood



# How lifting works

```
for(p <- Persons if p.name === "Stefan") yield p.name
```

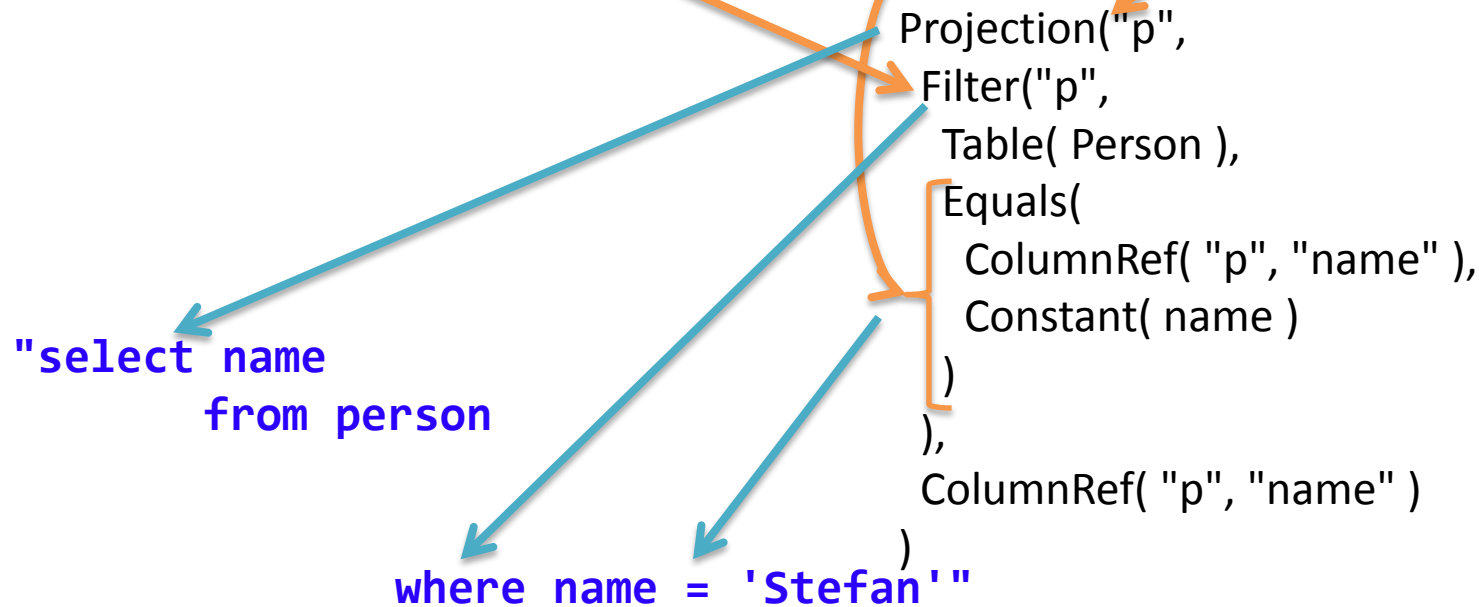
Scala desugaring



Column[String]

String (implicitly to Column[String])

```
Persons.withFilter(p=>p.name === "Stefan").map(p=>p.name)
```



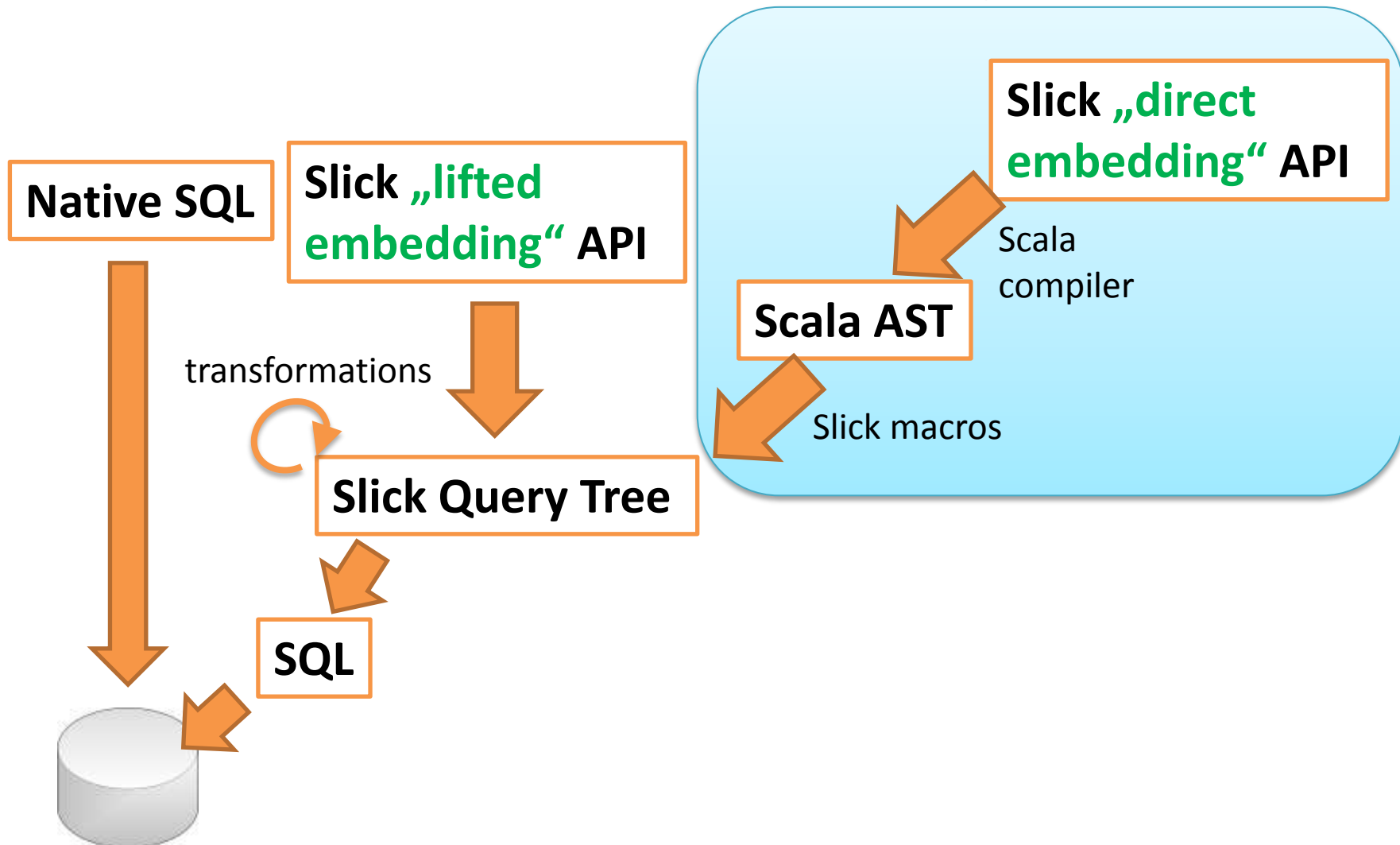
# Agenda

- Key Concepts
- Live Demo
- Under The Hood
- **Outlook**

# Beyond JDBC

- New back-end architecture
- MongoDB support
- Other NoSQL databases
- Enabling SQL-based non-JDBC drivers (e.g. SQLite on Android)
- Other data sources (e.g. Web Services)

# Direct Embedding



# Direct Embedding

- Real Scala (types, methods) using macros instead of emulation using lifting
  - no need to think about differences anymore
  - identical syntax
    - == instead of ===
    - if-else instead of Case.If-Else
    - ...
  - identical error messages
- Compile-time optimizations
- More compile-time checks



# Type Providers

- Based on *type macros*

```
object Coffees extends Table[(String, Int, Double)]("COFFEES") {
 def name = column[String]("NAME")
 def supID = column[Int]("SUP_ID")
 def price = column[Double]("PRICE")
 def * = name ~ supID ~ price
}
```

# Type Providers

- Based on *type macros*

```
object Coffees extends DBTable(
 "jdbc:h2:tcp://localhost/~ /coffeeShop",
 "COFFEES")
```

`type DBTable = macro ...`

```
val n = Coffees.
```

• name: Column[String] - Coffees  
• price: Column[Double] - Coffees  
• supID: Column[Int] - Coffees

Press 'Ctrl+Space' to show Template Proposals

# Nested Collections

- As seen in the **Scala Integrated Query** research prototype

```
for {
 s <- Suppliers
 c <- s.coffees
} yield (s, c)
```



Flat result set

# Nested Collections

- As seen in the **Scala Integrated Query** research prototype

```
for {
 s <- Suppliers
 val cs = s.coffees
} yield (s, cs)
```



Nested collection

- Multiple execution strategies are possible

 **Slick**.typesafe.com



@StefanZeiger

@typesafe