SLICK
Scala Language Integrated Connection Kit

Stefan Zeiger
Jan Christopher Vogt

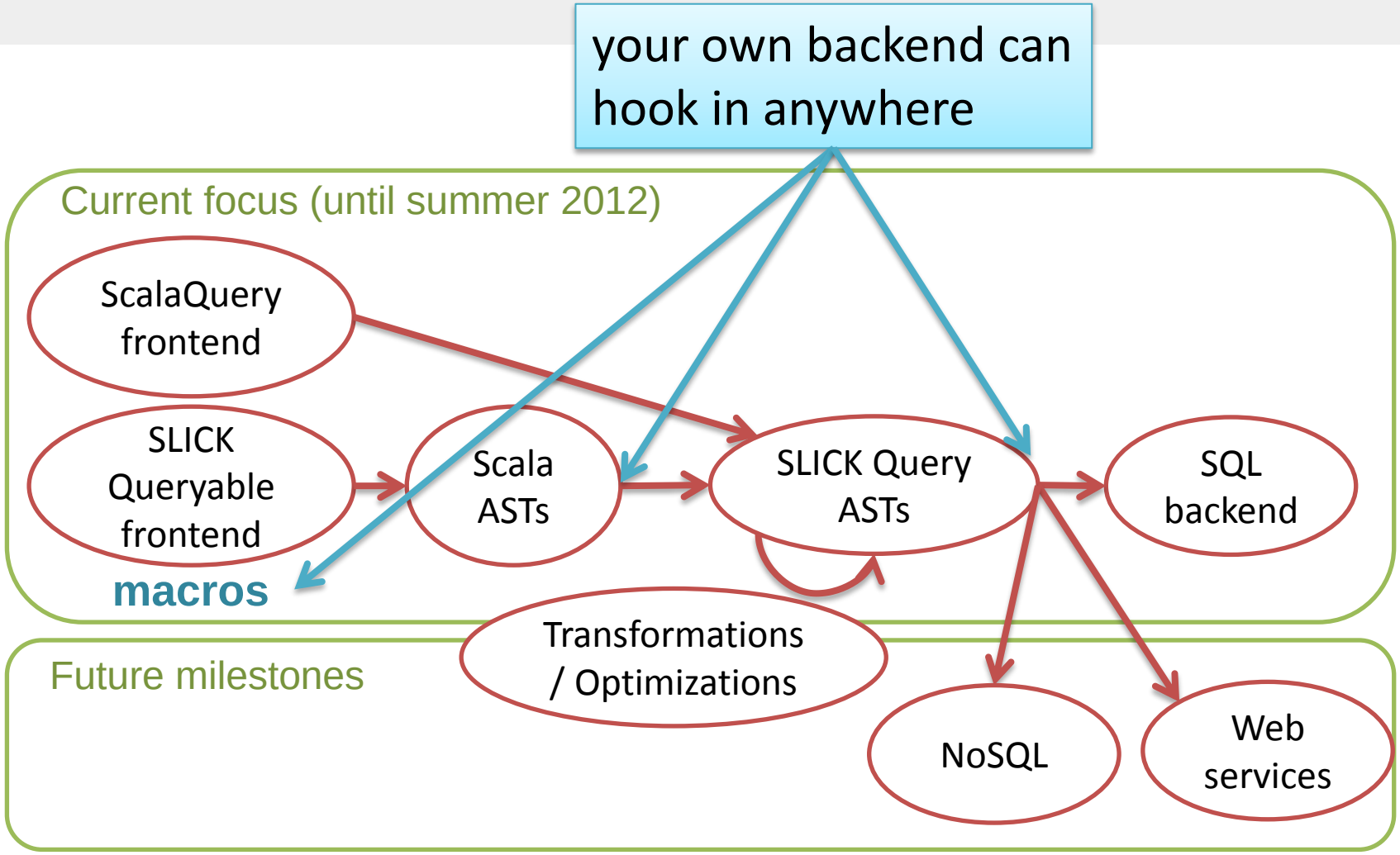
SLICK

- **Generic data query framework** (like LINQ)
- Integration of many backends: SQL, NoSQL, …

- Based on ScalaQuery code and experiences from Scala Integrated Query

- **`scala.slick.Queryable`**:
  an alternative, easier (yet limited) query frontend

Architecture



Queries on Collections

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


val coffees = List(
  Coffee("Colombian",         101, 7.99),
  Coffee("Colombian_Decaf",   101, 8.99),
  Coffee("French_Roast_Decaf", 49, 9.99)
)

val l = for {
  c <- coffees if c.supID == 101
} yield (c.name, c.price)

l.foreach { case (n, p) => println(n + ": " + p) }
```

Scala Collections

Queries using ScalaQuery frontend

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

Coffees.insertAll(
        ("Colombian",         101, 7.99),
        ("Colombian_Decaf",   101, 8.99),
        ("French_Roast_Decaf", 49, 9.99)
)

val q = for {
  c <- Coffees if c.supID === 101
} yield (c.name, c.price)

q.foreach { case (n, p) => println(n + ": " + p) }
```

ScalaQuery

Queries using Queryable frontend (prototype, work in progress)

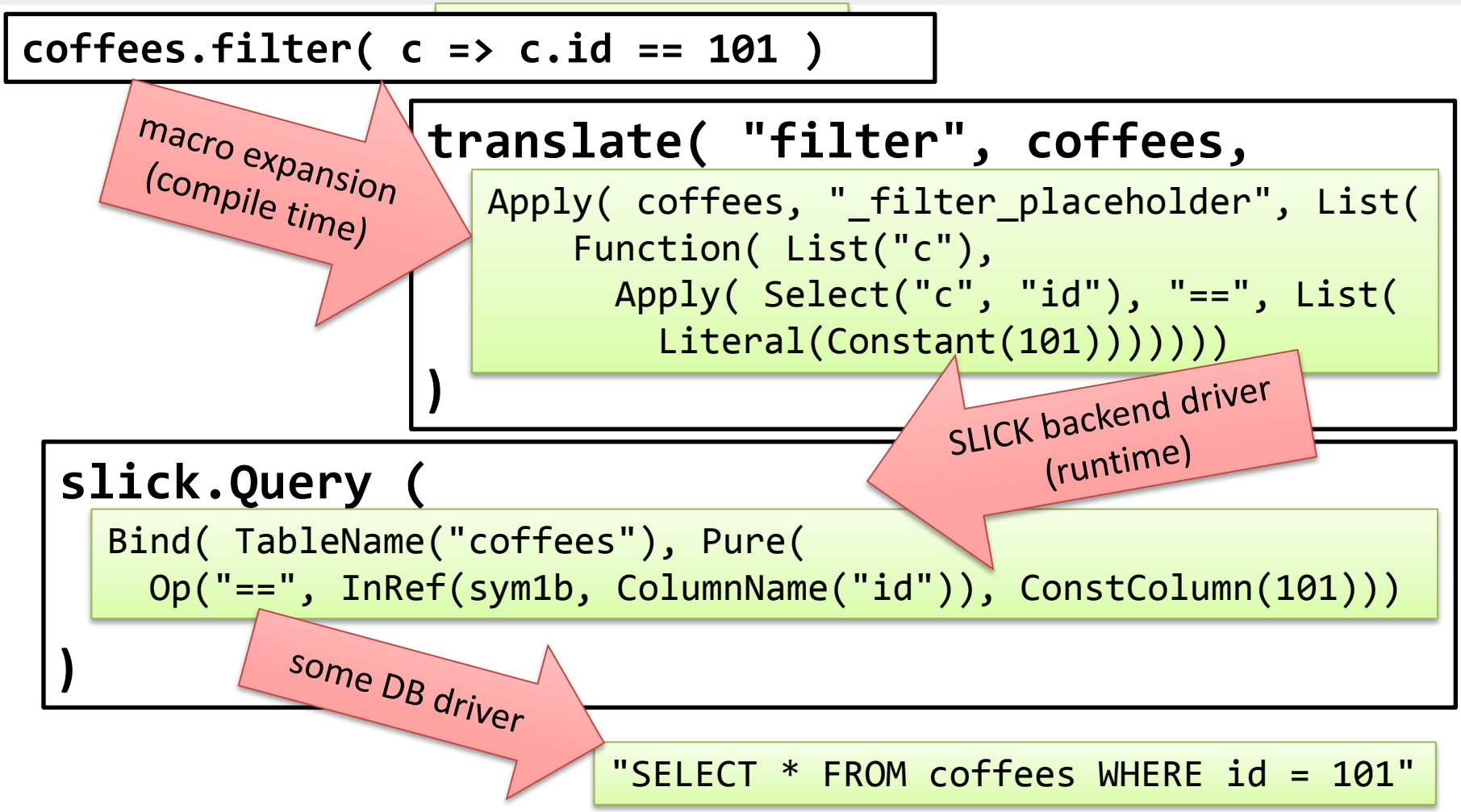
```
@table("COFFEES")
case class Coffee(
  @column("COF_NAME")  name:  String,
  @column("SUP_ID") supID: Int,
  @column("PRICE") price: Double
)
val backend = SlickBackend(MySqlBackend("dsn://..."))
val coffees = Queryable[Coffee](backend)
coffees ++= List(
  Coffee("Colombian",          101, 7.99),
  Coffee("Colombian_Decaf",    101, 8.99),
  Coffee("French_Roast_Decaf", 49, 9.99)
)

val l = for {
  c <- coffees if c.supID == 101
} yield (c.name, c.price)

l.foreach { case (n, p) => println(n + ": " + p) }
```

currently annotations, later different ways

SLICK Queryable

macros

Queryable – internal steps



Queryable transparent execution

- **coffees.map(…)**
  lazy: returns a Queryable (e.g. map)
- **coffees.length**
  strict (executes): returns a scalar value
- **coffees.toList**
  strict (executes)
- **Queryable{**
  (coffees.count, coffees.map(_.price).sum) }
  lazy: wrapped in Queryable scope

completely transparent
=> easy to use

Comparison

ScalaQuery frontend

- based on implicits / overloading
- Rep[T] types
- sometimes confusing errors
- stronger type-safety
- fully compositional

Queryable frontend

- based on macros
- closer to Scala collections
- ordinary Scala types
- better error messages
- weaker type-safety
- partially compositional
- Future:
  - stronger type-safety
  - better compositionality

ScalaQuery AST changes

- **AST nodes in `scala.slick.ast`**
- **No dependency on ScalaQuery front-end** – ASTs are easy to build manually or with other generators (e.g. the Queryable front-end)
- **Tree transformers** bring the AST into the proper shape for code generation
- Leads to **simpler `QueryBuilder` implementations** for SQL code generation

ScalaQuery front-end changes

- **Shapes of record („row“) types are encoded in a Shape typeclass** to support more complex shapes like nested tuples
- **Closer to Scala collection semantics** (e.g. .sortBy/.groupBy vs SQL) for **better compositionality**
- **Query result types encoded in queries** – provides uniform execution semantics for scalar, record and collection valued queries

Release Schedule

https://www.assembla.com/spaces/typesafe-slick/milestones

- Initial release: **Summer of 2012**
- Semi-annual milestones over the next 2 years

Type-Generating Macros

- Like *Type Providers* in .NET but based on macros instead of compiler plug-ins

```
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
}
```

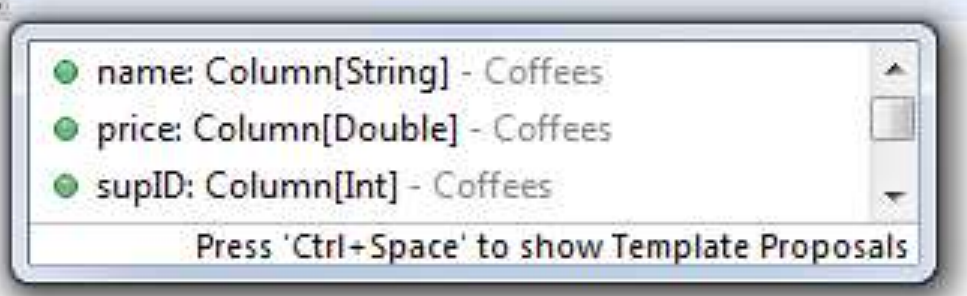
Type-Generating Macros

- Like *Type Providers* in .NET but based on macros instead of compiler plug-ins

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

```
type DBTable = macro ...
```

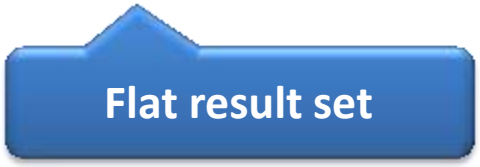
```
val n = Coffees.
```



Nested Collections

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

```
for {
  s <- Suppliers.sortBy(_.id)
  c <- s.coffees if c.price < 9.0
} yield ((s.id, s.name), c)
```



Nested Collections

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

```
for {
  s <- Suppliers.sortBy(_.id)
  val cs = s.coffees.filter(_.price < 9.0)
} yield ((s.id, s.name), cs)
```

- Multiple execution strategies are possible

Other Features

- Support **more relational databases**
- Extend semantics to cover **NoSQL databases** and other data sources
- **Optimizations**
- **`Queryable`**
  - **Stronger type-safety**
  - **Better compositionality** across compilation units
- Lift **`Queryable`** values to **`Query`**

Resources

- SLICK project plan & bug tracker: https://www.assembla.com/spaces/typesafe-slick/
- New macro-based front-end: https://github.com/cvogt/slick/
- ScalaQuery: http://scalaquery.org
- Refactored ScalaQuery codebase for SLICK: https://github.com/szeiger/scala-query/tree/new-ast

Thank You!

Jan Christopher Vogt

- christopher.vogt@epfl.ch
- http://code.google.com/p/scala-integrated-query/

Stefan Zeiger

- Blog: http://szeiger.de
- ScalaQuery: http://scalaquery.org