
# Scala Macros

**Eugene Burmako**
**Jan Christopher Vogt**

London, 2012

# Scala Macros

Empower developers to
extend the compiler and stay sane!

This enables compile-time:

- checks
- processing
- AST transformations
- code generation
- shipping of ASTs to runtime

# Implementation

Compile-time metaprogramming has long existed in Lisp, so it should be easy to implement, right?

# Trickiness

Homoiconicity for a language with syntax and types is hard.

Being non-hygienic is evil, being hygienic imposes a complexity tax on the spec.

Need quasiquotations to make AST manipulations bearable. Yet another concept.

# Beauty

Martin Odersky was suspicious:

“And, we'd need to be convinced that it is beautifully simple, or it won't go into Scala”.

Here is what convinced Martin:

# Cats

# The essence

Scala reflection provides a slice of the compiler's cake. The infrastructure is already there.

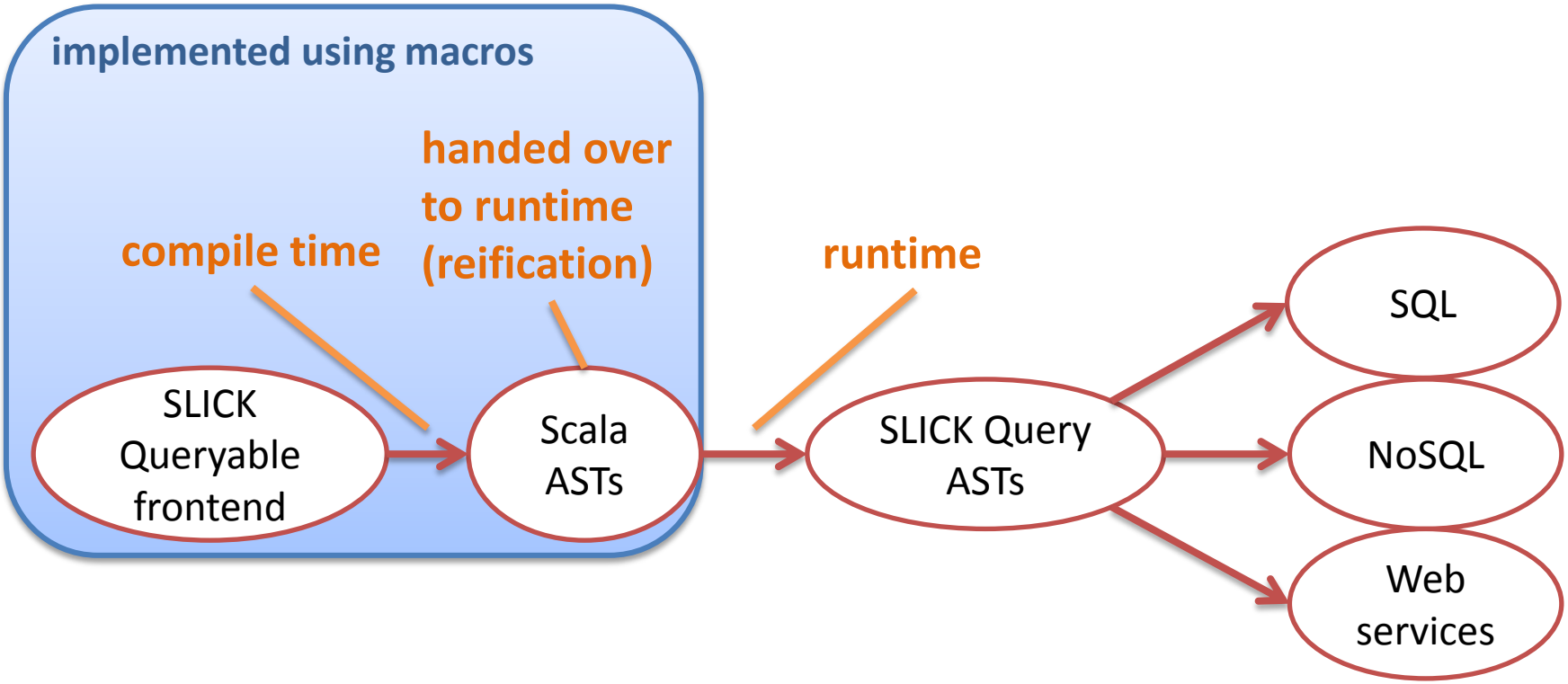
Macros are (Tree*, Type*) => Tree functions. Just as simple as that.

Hygiene itself is implemented by a macro. Hence we stay minimalistic and flexible.

# USE CASE: SLICK

# SLICK overview

```
val coffees = Queryable[Coffee]( /* db connection */ )
coffees       .filter( c => c.id == 101 )
```



# Macros in SLICK

```
val coffees = Queryable[Coffee]( /* db connection */ )
coffees      .filter( c => c.id == 101 )
```

implemented as a macro that works on argument AST at compile time

```
def filter[T]( predicate: T => Boolean ) : Queryable[T] =
                                  macro QueryableMacros.filter[T]

object QueryableMacros{
  def filter[T:c.TypeTag] (c: scala.reflect.makro.Context)
                 (predicate : c.mirror.Expr[T => Boolean]) = …
}
```

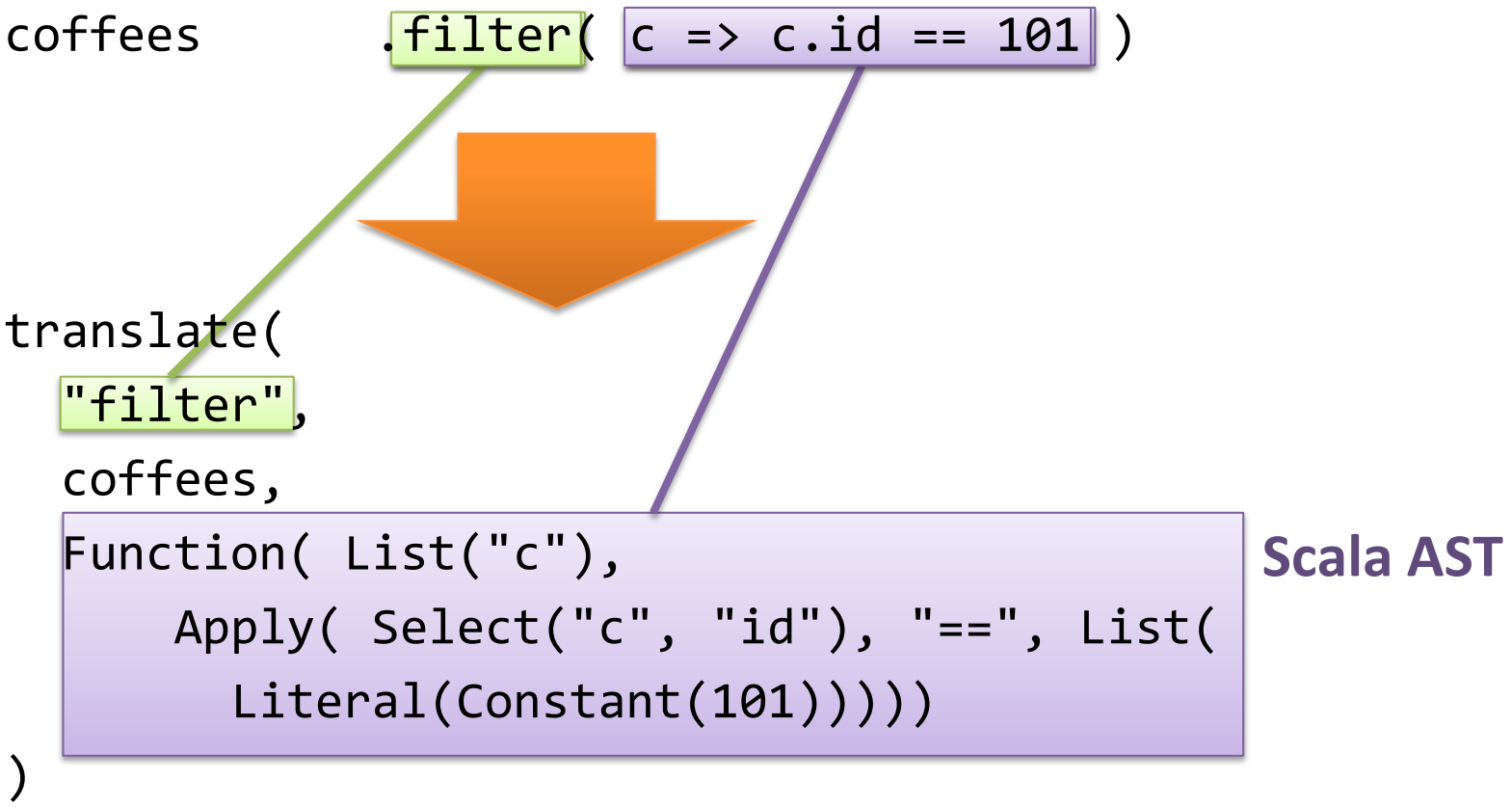
# A macro body

```
object QueryableMacros{

  // macro implementation signature
  def filter[T:c.TypeTag] (c: scala.reflect.makro.Context)
                (predicate: c.mirror.Expr[T => Boolean]) = {

  // reify tree (ship to runtime)

  val reifiedPredicate =
      c.mirror.Expr[ reflect.mirror.Expr[T => Boolean] ](
        c.reifyTree( c.reflectMirrorPrefix, projection.tree ) )

  // splice into runtime call
  c.reify{ translate("filter", c.prefix.eval, reifiedPredicate.eval ) }
}}
```

splicing

inlined at the call site (macro expansion),
translates queries further at runtime, e.g. to SQL

# Resulting macro expansion




**That are macros in SLICK**

# Applications in the compiler

- Removed a compiler phase (LiftCode)
- Made a 80% solution (manifests) a 99% solution (type tags)
- Manifests à la carte – no longer hardcoded in Implicits.scala, it‘s just several macros (you can write your own implementations!)
- Ideas: SourceLocations, SourceContexts, static requirements on the compiler and the compilation environment

# Applications in the wild

- SLICK
- Macrocosm
- Scalatex
- Expecty
- Scalaxy
- Ideas: procedure typing with arrows, lens generation, ACP DSL, zero-overhead mocks

# Future work

- Debugging generated code
- Untyped macros
- Type macros
- Macro annotations
- Replacing the compiler with a macro

# Future work

- Debugging generated code
- Untyped macros
- Type macros
- Macro annotations
- ~~Replacing the compiler with a macro~~

Come on, just kidding about the last one!

# Thank you!

Questions and answers

www.scalamacros.org