Why ScalaQL

- Impedance mismatch between the relational model and the paradigm of PL (Scala).
- Embedded SQL statements are error-prone
- Cannot be checked at compile time

For-comprehension

- for-comprehensions are a natural syntactic device for representing declarative queries
- Example:
  ```scala
  val underAge = for (
    p <- Person
    c <- Company
    if p.company == c
    if p.age < 14
  ) yield p
  ```
- SQL correspondence
  ```sql
  SELECT p.* FROM people p JOIN companies c ON p.company_id = c.id
  WHERE p.age < 14
  ```

For-comprehension

- Each for-comprehension is translated into series of calls on `Map`, `flatMap` and `filter` methods.
- Objects that for-comprehension targets on should implement those methods.

Language Features

- Deferred evaluation
  - Not evaluate until the query is used as sequence
  - Represented by AST
- Generate items of new data type in the fly
  - Query[T] where T could be dynamical generated by `new (...)`
  - Type inference

Compilation

- Annotation
- LINQ to Scala
- Scala to Ferry

Architecture
Annotation

- ScaleQL recognizes two types of queries
  - Embedded LINQ queries with an @LINQAnn annotation
  - Scala queries, tagged with @Persistent annotation

LINQ to Scala

- Convert LINQ query to string with annotations
- Parse the string and generate the Scala for-comprehensions
- Tag with @Persistent

Scala to Ferry

- Nested Scala tuple → flattened Ferry tuple
- Type checking
- AST representing Scala query will used for translation