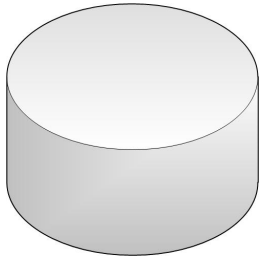
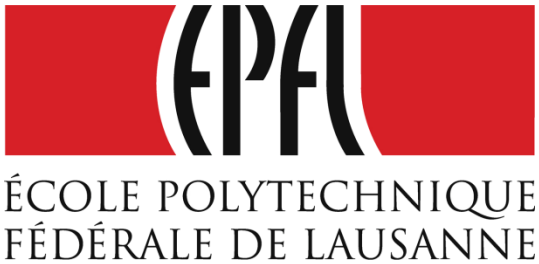


Scala Integrated Query

Christopher Vogt



Scala Integrated Query

short SIQ

- a type safe database query library

  like Scala Query *by Stefan Zeiger* and SQueryl

- compiles a subset of Scala to SQL

Feature Comparison

	SQL	Scala Query	SQueryl	Microsoft's LINQ-to-SQL on .NET	**Scala Integrated Query**
Queries	✔	✔	✔	✔	✔
Type Safety		✔	✔	✔	✔
List Order					✔
Avalanche Safe Nesting					✔
…	…	…	…	…	…

AVALANCHE SAFE NESTING

Example (in pseudo code)

for nation <- nations

(nation, customers of this nation)

Result: List[(Nation, List[Customer])]

```
List(
   ("GERMANY", List("Martin","Chris") ),
   ("ARGENTINA", List("Miguel",...) )
)
```

How many SQL Queries?

for nation <- nations

(nation, customers of this nation)

Depends on your code:

	SQL	Scala Query	SQueryl	Microsoft's LINQ-to-SQL on .NET	**Scala Integrated Query**
straight-forward	**n + 1**	**n + 1**	**n + 1**	**n + 1**	**2**
manually tweaked	**2**	**2**	**2**	**2**	

Avalanche Safe

n = nations.length

Example: Scala Query

```
val nations = ( for( n <- nation ) yield n ).list

for( n <- nations ) yield {
    val cs =
        ( for( c <- customer;
            if c.nationkey === n.nationkey
          ) yield c ).list
    (n, cs)
} // : List[ ( Nation, List[Customer]  ) ]
```

SQL Mapping in Scala Query

JVM

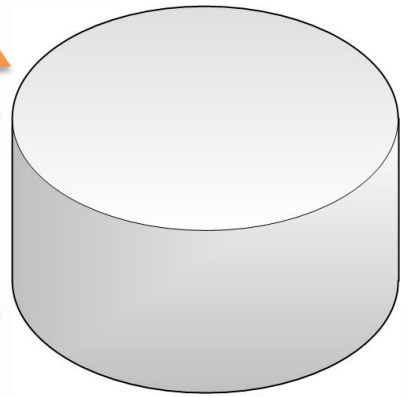
DBMS

1 SQL Query
Example: 200 results

```
val nations = ( for( n <- nation ) yield n ).list

for( n <- nations ) yield {
    val cs =
        ( for( c <- customer;
            if c.nationkey === n.nationkey
          ) yield c
        ).list
    (n, cs)
}
```

200 SQL queries
An „avalanche“



SIQ does this better

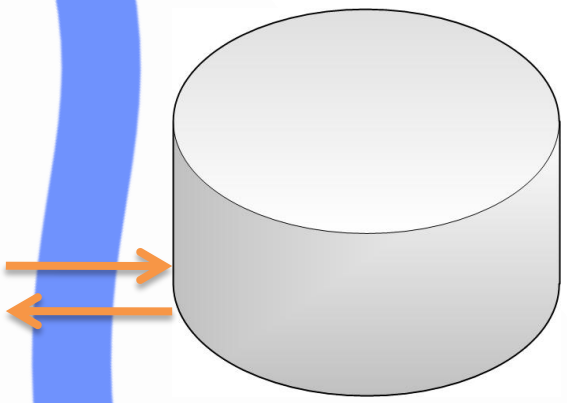
Scala Integrated Query: No Avalanche

JVM

DBMS

```
val nations = for( n <- nation ) yield n

(for( n <- nations ) yield {
    val cs =
          for( c <- customer;
                if c.nationkey == n.nationkey
          ) yield c
    (n, cs)
})
.fromdb
```



2 SQL queries
No „avalanche“

Mapped to SQL as a whole

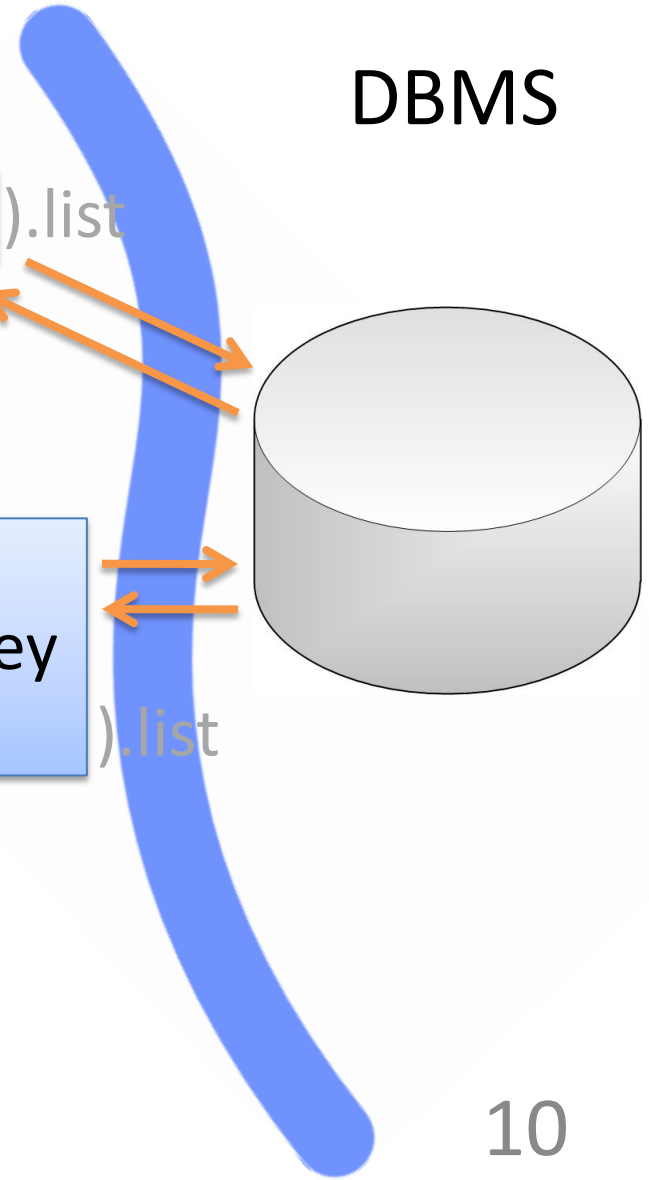
SQL Mapping in Scala Query

JVM

DBMS

```
val nations = ( for( n <- nation ) yield n ).list

for( n <- nations ) yield {
    val cs =
        ( for( c <- customer;
            if c.nationkey === n.nationkey
          ) yield c ).list
    (n, cs)
}
```

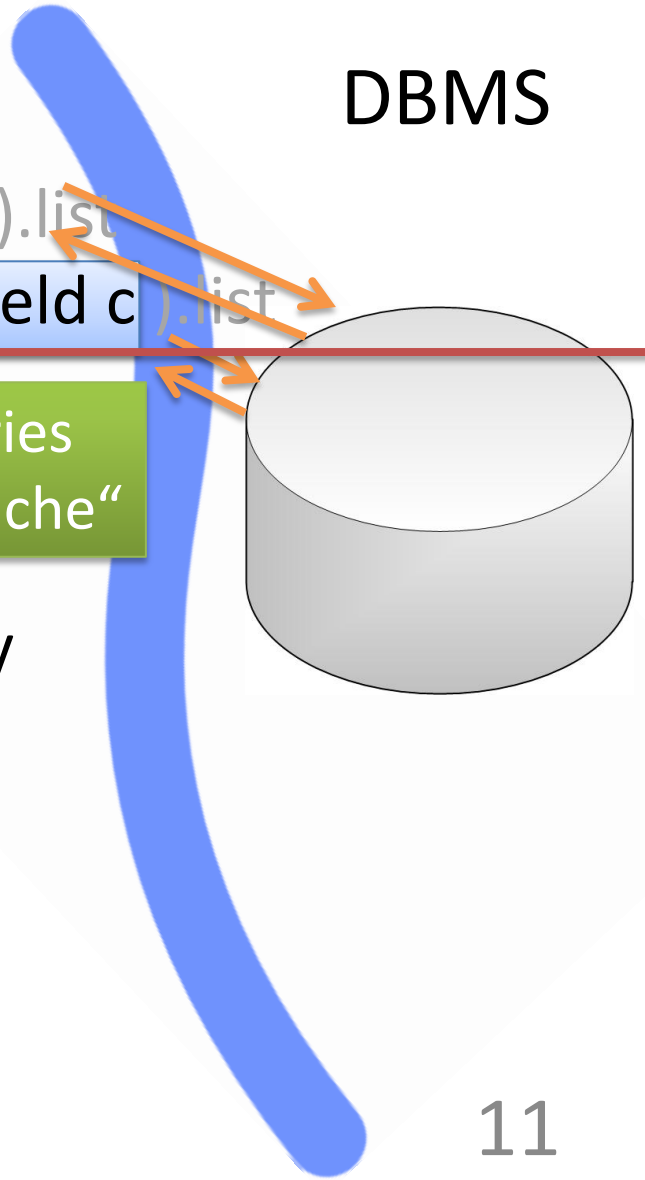


Avalanche avoidance in Scala Query

JVM

DBMS

```
val nations = ( for( n <- nation ) yield n ).list
val customers = ( for( c <- customer ) yield c ).list
for( n <- nations ) yield {
    val cs =
        for( c <- customers;
            if c.nationkey == n.nationkey
        ) yield c
    (n, cs)
}
```



2 SQL queries
No „avalanche“

In SIQ you get this for free!

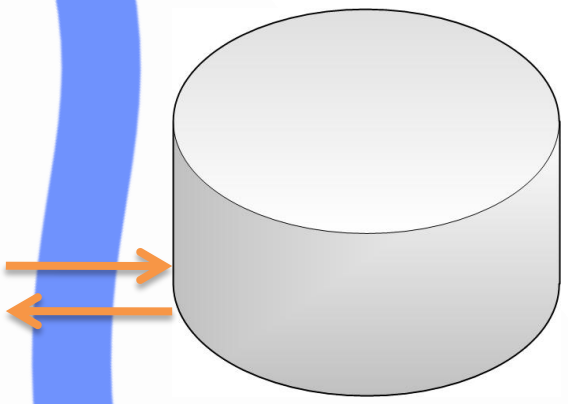
Scala Integrated Query: No Avalanche

JVM

DBMS

```
val nations = for( n <- nation ) yield n

(for( n <- nations ) yield {
    val cs =
        for( c <- customer;
            if c.nationkey == n.nationkey
        ) yield c
    (n, cs)
})
.fromdb
```



Avalanche Safety for free

SIQ: Let‘s simplify

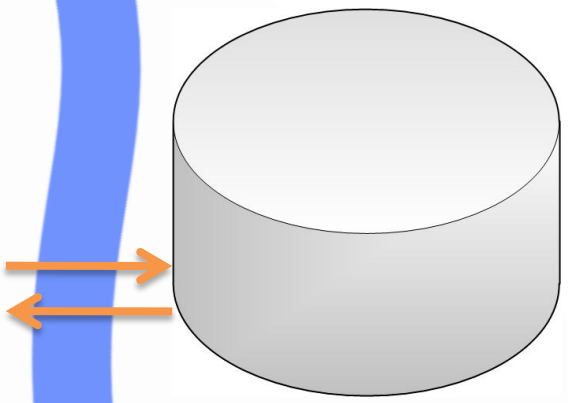
JVM

DBMS

~~val nations = for(n <- nation) yield n~~

```
(for( n <- nation~~s~~ ) yield {
    val cs =
          for( c <- customer;
                if c.nationkey == n.nationkey
          ) yield c
    (n, cs)
})
```

.fromdb

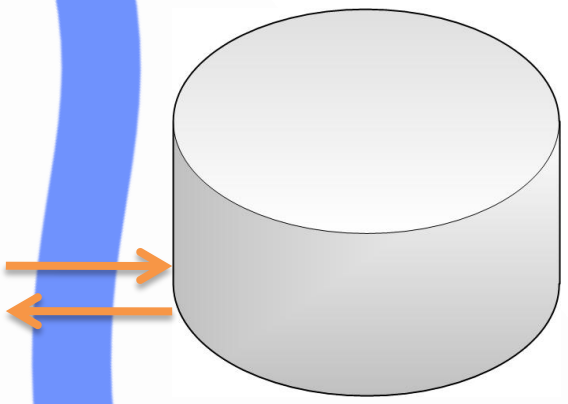


SIQ: Let‘s simplify

JVM

DBMS

```
(for( n <- nation ) yield {
    val cs =
          for( c <- customer;
                if c.nationkey == n.nationkey
          ) yield c
    (n, cs)
})
.fromdb
```



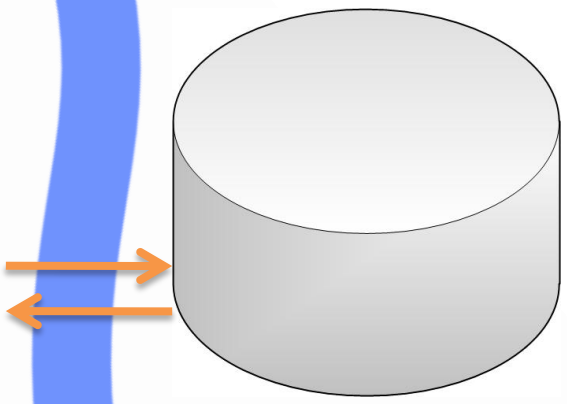
SIQ: Let‘s simplify

JVM

DBMS

```
nation.map( n =>
    ( n,  customer.withFilter(
            _.nationkey == n.nationkey
    ))
)
```

.fromdb



SIQ: Query Decomposition

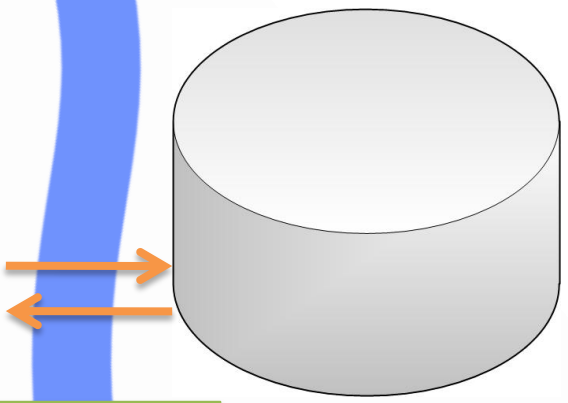
JVM

DBMS

```
def customers( nationkey : Rep[Int] ) =
    customer.withFilter(
            _.nationkey == nationkey
    )

nation.map( n =>
     (n, customers(n.nationkey) )
)
```

.fromdb



still only
2 SQL queries

SIQ: Data Model

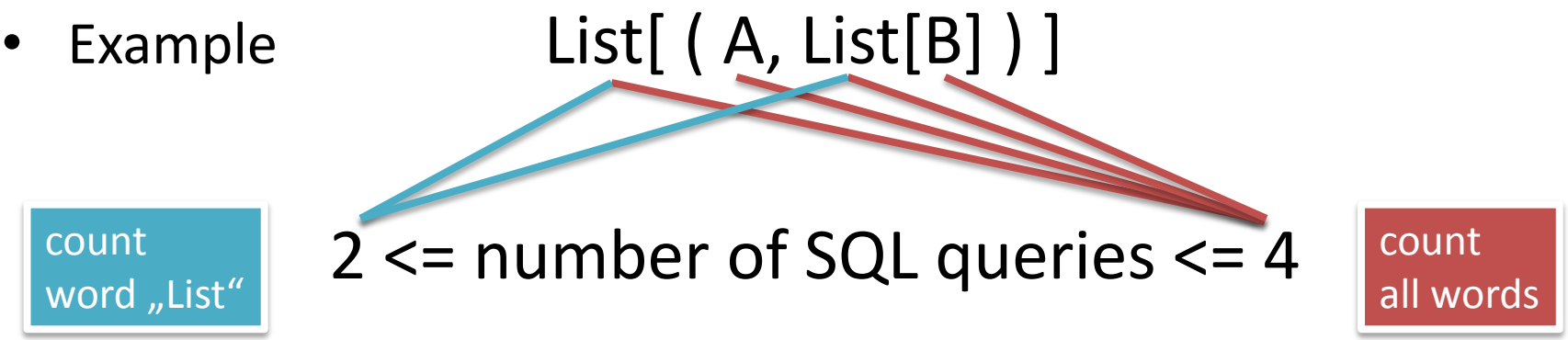
- Seen so far
  - List[(Nation, List[Customer])]

- Supported: **Arbitrarily nested Lists and Tuples**

- you can mix database tables and Iterables

How many SQL queries exactly?

- Depends only on **result type**
- **Lower Bound**
  Number of list constructors
- **Upper Bound**
  Number of all type constructors excluding tuples
- Example



HOW DOES AVALANCHE SAFETY WORK?

SIQ implements Ferry

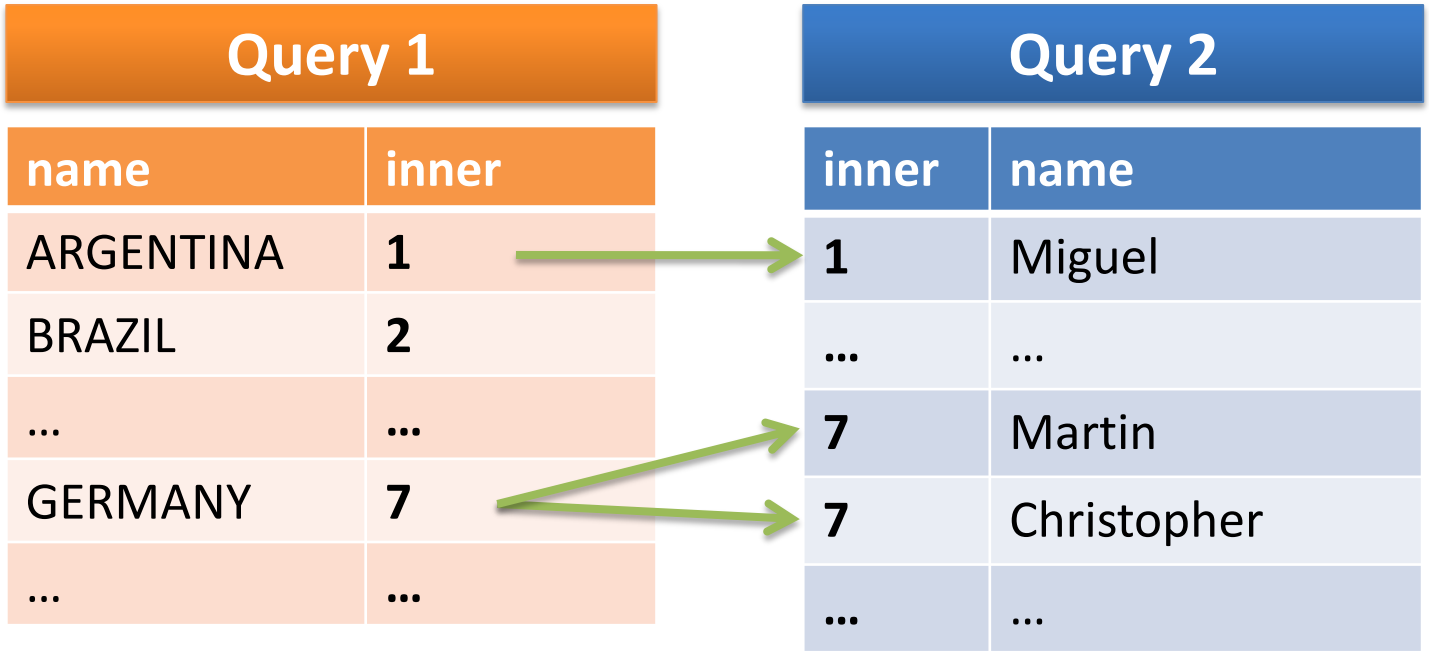
- nested comprehensions → relational algebra
- theory developed by Tom Schreiber
  University of Tübingen
- (Great stuff, Tom ☺!)

Eberhard Karls
Universität
Tübingen

Ferry SQL Mapping

for nation <- nations

(nation, customers of this nation)

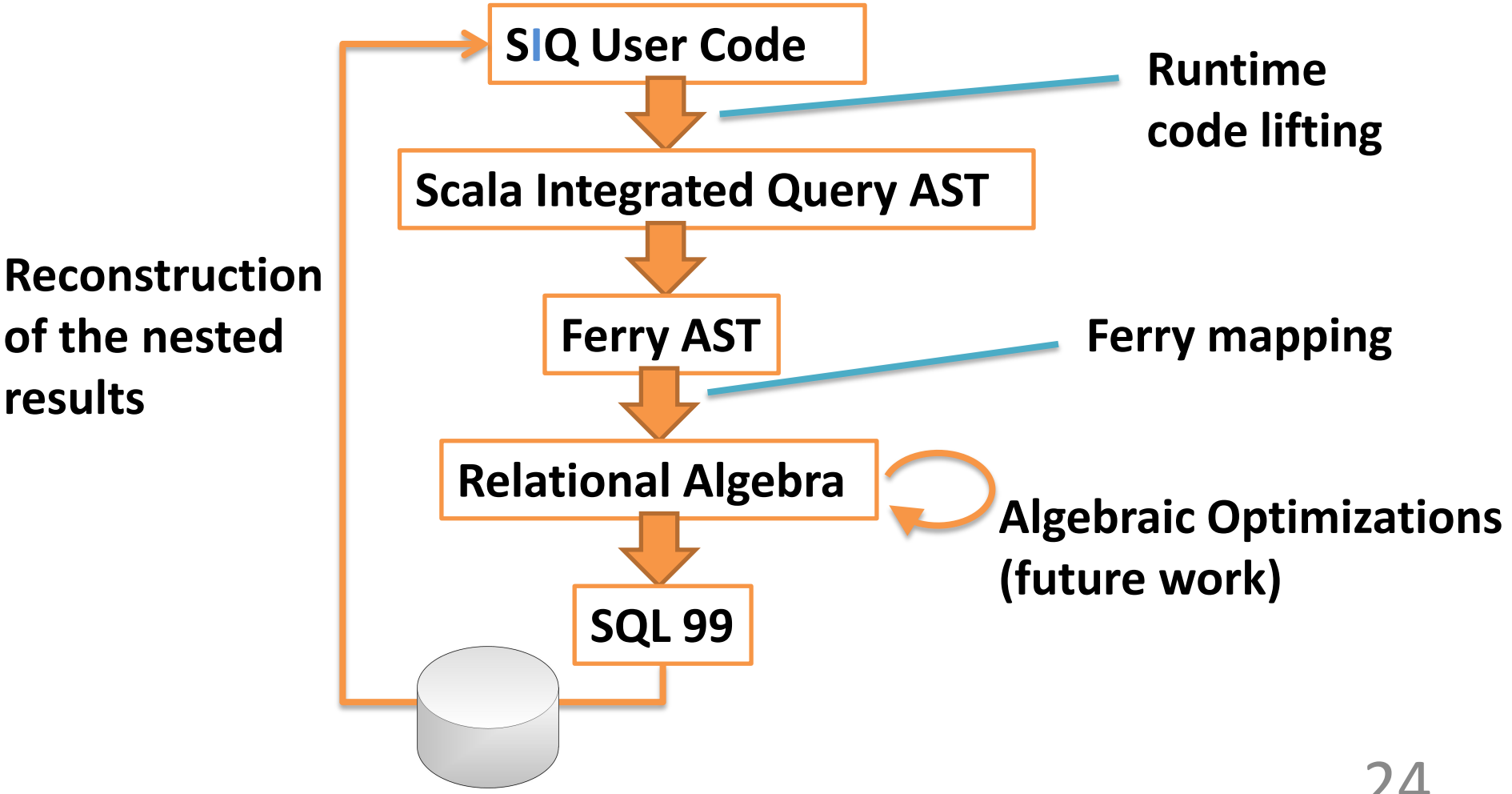


STAGING

SIQ Implementation

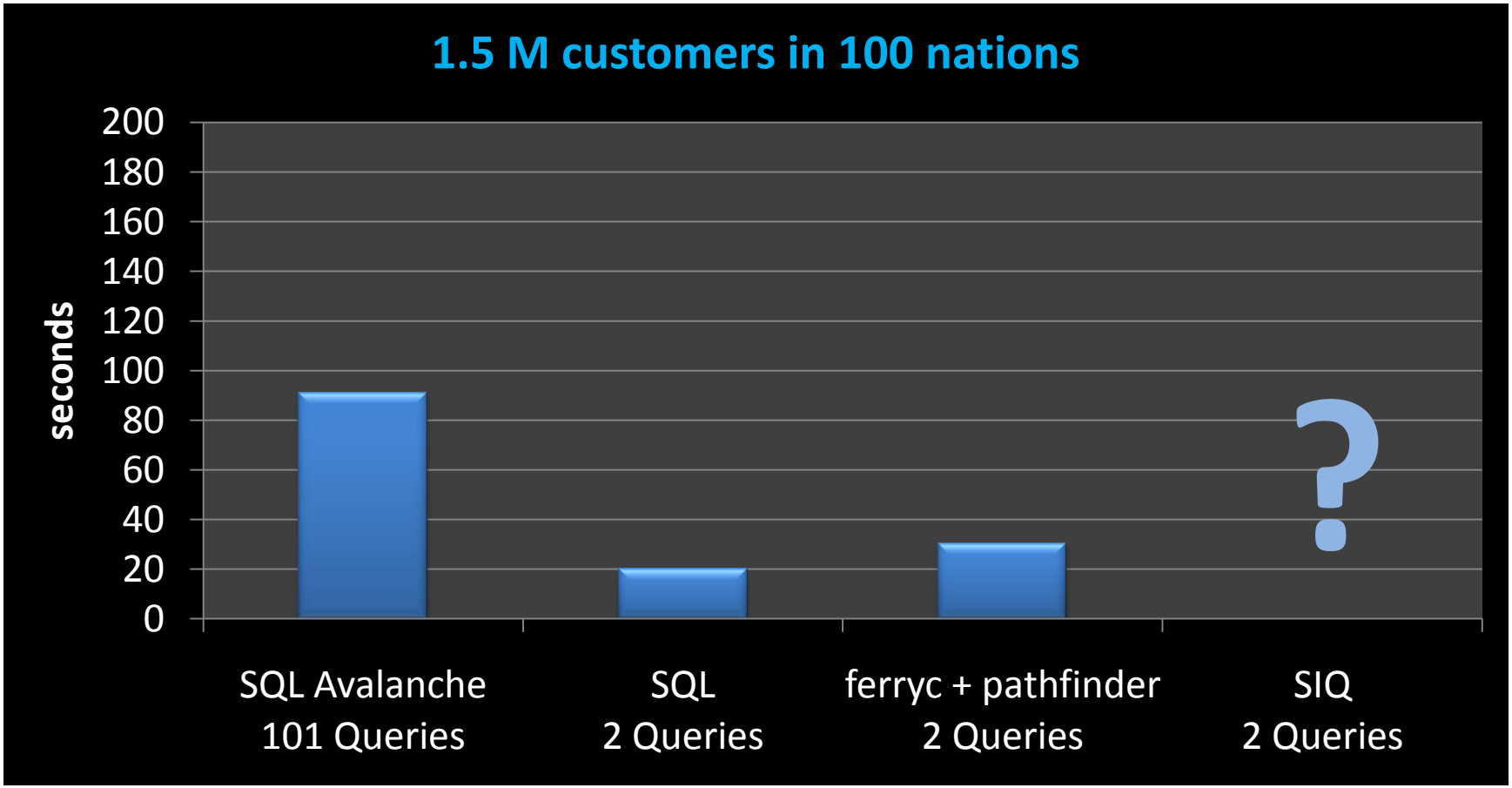
- **Staging** (EPFL / Stanford)
  a technique used in
  - OptiML
  - Delite
  - Scala Integrated Query
  - …
- light-weight modular staging framework
- **Scala Virtualized**

Staging in Scala Integrated Query



BENCHMARK

SQL execution time



10 GB TCP-H, Average of 10 runs, Postgres,Windows 7 32bit, Intel Core i5 M 580, 4GB RAM

Ferry list order requires SQL ROW_NUMBER, so for comparability used in all queries here

Supported Scala Subset

currently	coming up
map, flatMap, withFilter, length, + - * / %, distinct, flatten, ++, == >= <= > <, +, && ||, unzip, orderBy, Tuple1-23, zip, groupBy	head, db functions, max, min, take, apply, mkString, drop, zipWithIndex, sum, avg, more ...

Future Work

- Make it production ready
  - I will be joining Typesafe in Autumn
- Algebraic optimizations
  - collaboration with University of Tübingen
  - taken from their „Pathfinder“ project
  - with help of Tom Schreiber, the inventor of Ferry
- Collaboration with Stefan Zeiger / Scala Query

Acknowledgements

- **Prof. Bernhard Rumpe**
- **Prof. Martin Odersky**
- **Miguel Garcia**
- **Tiark Rompf**
- **Tom Schreiber**
- **Prof. Torsten Grust**

check out the prototype at

http://code.google.com/p/scala-integrated-query/

Other Ferry implementations exist for
.NET/LINQ, Haskell, Ruby, Links

Thank you!

http://code.google.com/p/scala-integrated-query/

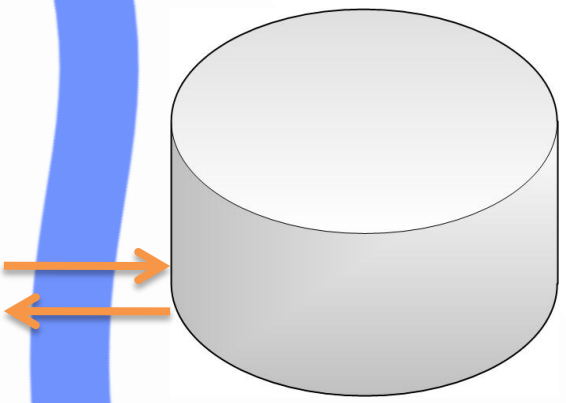
SIQ: Element access before execution

JVM

DBMS

```
def customers( nationkey : Rep[Int] ) =
    customer.withFilter(
            _.nationkey == nationkey
    )

nation.map( n =>
     (n, customers(n.nationkey) )
).withFilter( _._1.name == „EGYPT“ )
  .map( _._2)
.fromdb
```

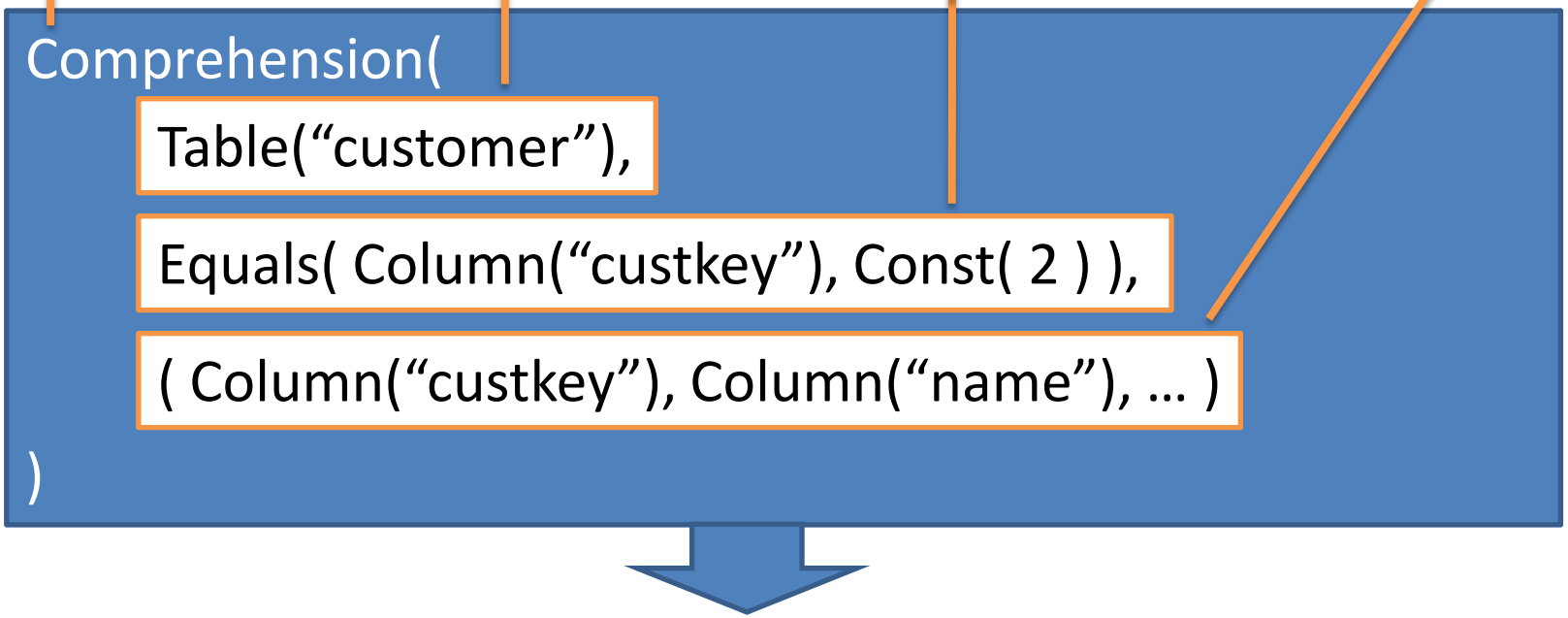


only 1 SQL query in total

BEHIND THE SCENES
- CODE LIFTING

Code lifting - internal view (simplified)

- code-lifting, query returns AST
- for(c <- customer; if c.custkey == 2) yield c



- SELECT * FROM customer WHERE custkey = 2

Code lifting – How lifting works

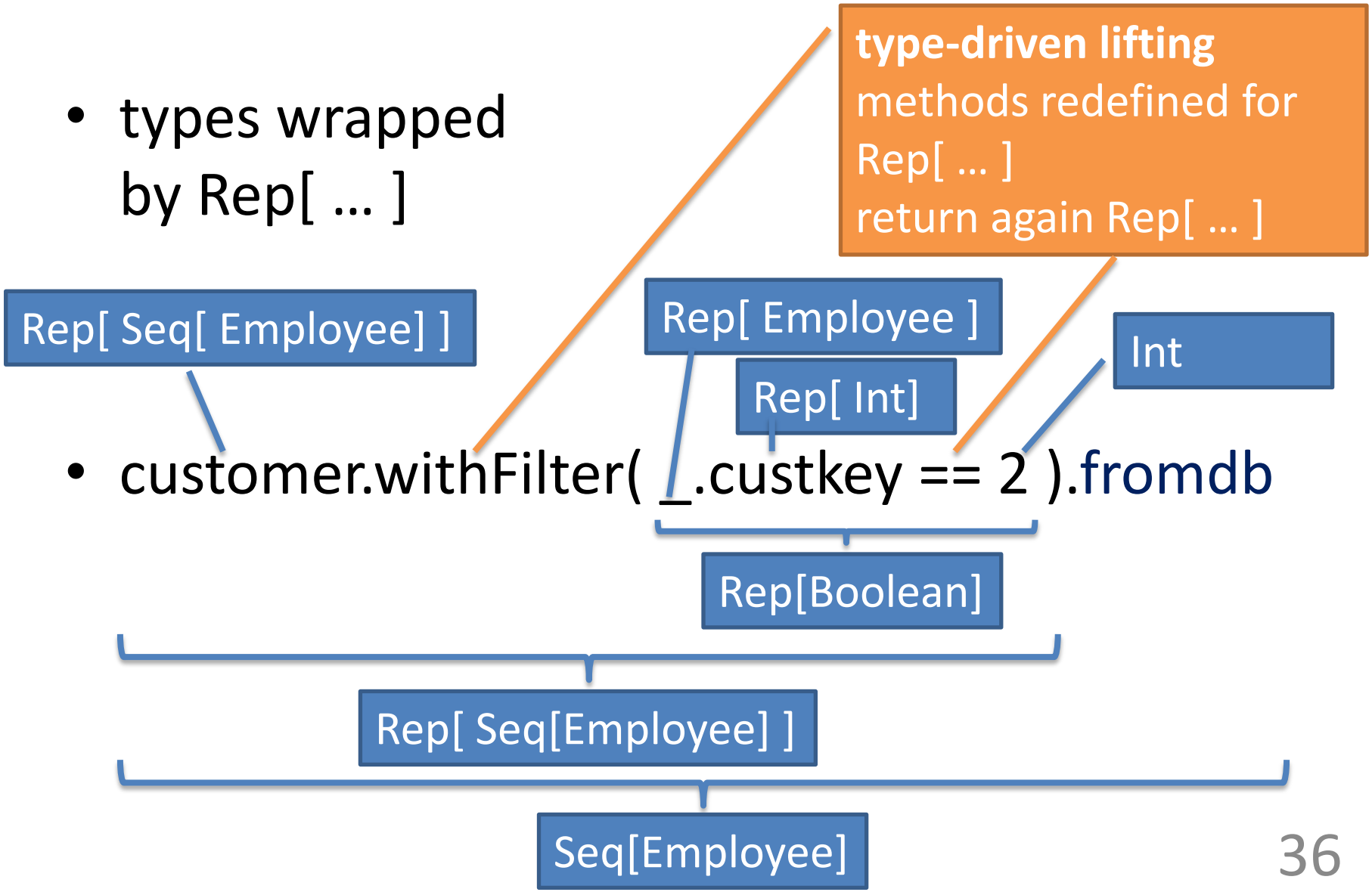
- implicit conversions
- overloading
  - methods: flatten, head, +, -, …
  - in Scala Virtualized also: ==, if-then-else, …
  - for-comprehensions: map, flatMap, withFilter

```
for( c <- customer; if c.custkey == 2 ) yield c
```

```
customer.withFilter( c => c.custkey == 2 )
```

Code lifting - user view

- types wrapped by Rep[…]



BEHIND THE SCENES - FERRY

Ferry Mapping

for nation <- nations

(nation, customers of this nation)

- done in 2 steps (all in the database)
  1. **loop lifting**: for each nation ALL customers
  2. **filter all at once**: keep only relevant customers

Ferry Mapping

for nation <- nations

(nation, customers of this nation)

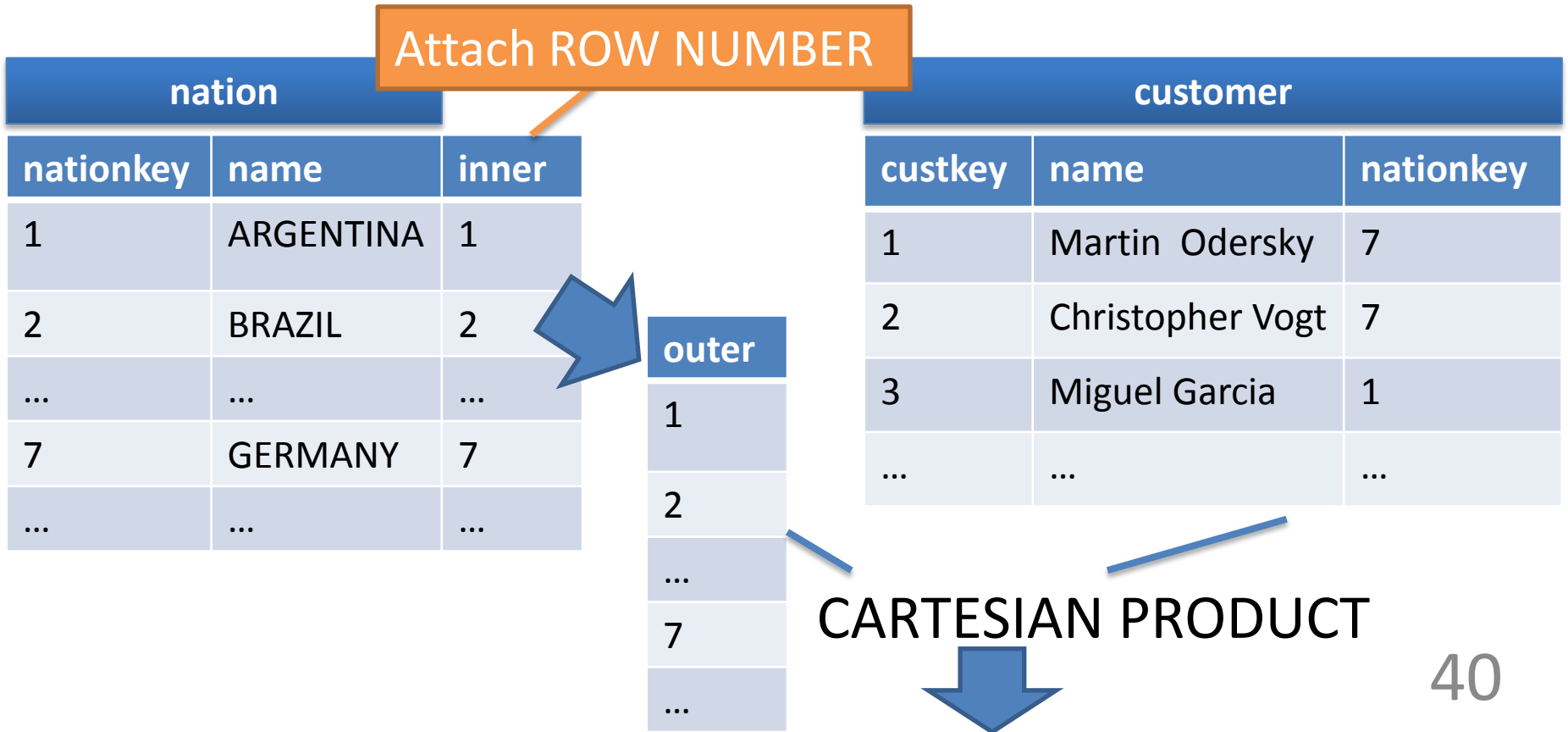
nation	
nationkey	**name**
1	ARGENTINA
2	BRAZIL
…	…
7	GERMANY
…	…

customer		
custkey	**name**	**nationkey**
1	Martin Odersky	7
2	Christopher Vogt	7
3	Miguel Garcia	1
…	…	…

Step 1: Loop Lifting

for each nation ALL customers

for(n <- nation) yield customer

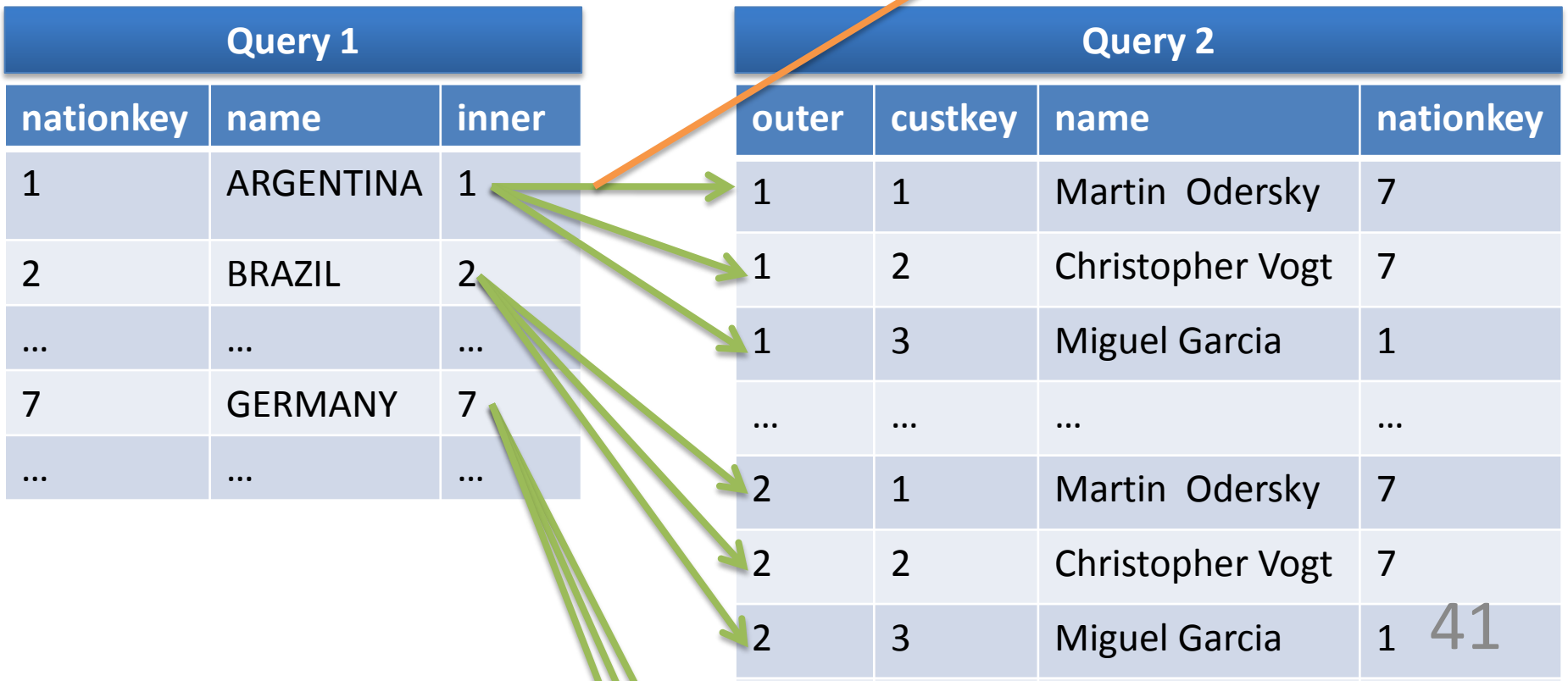


After Loop Lifting: Nesting

for each nation ALL customers

for(n <- nation) yield customer

can be used to construct nested results



Query 1

nationkey	name	inner
1	ARGENTINA	1
2	BRAZIL	2
…	…	…
7	GERMANY	7
…	…	…

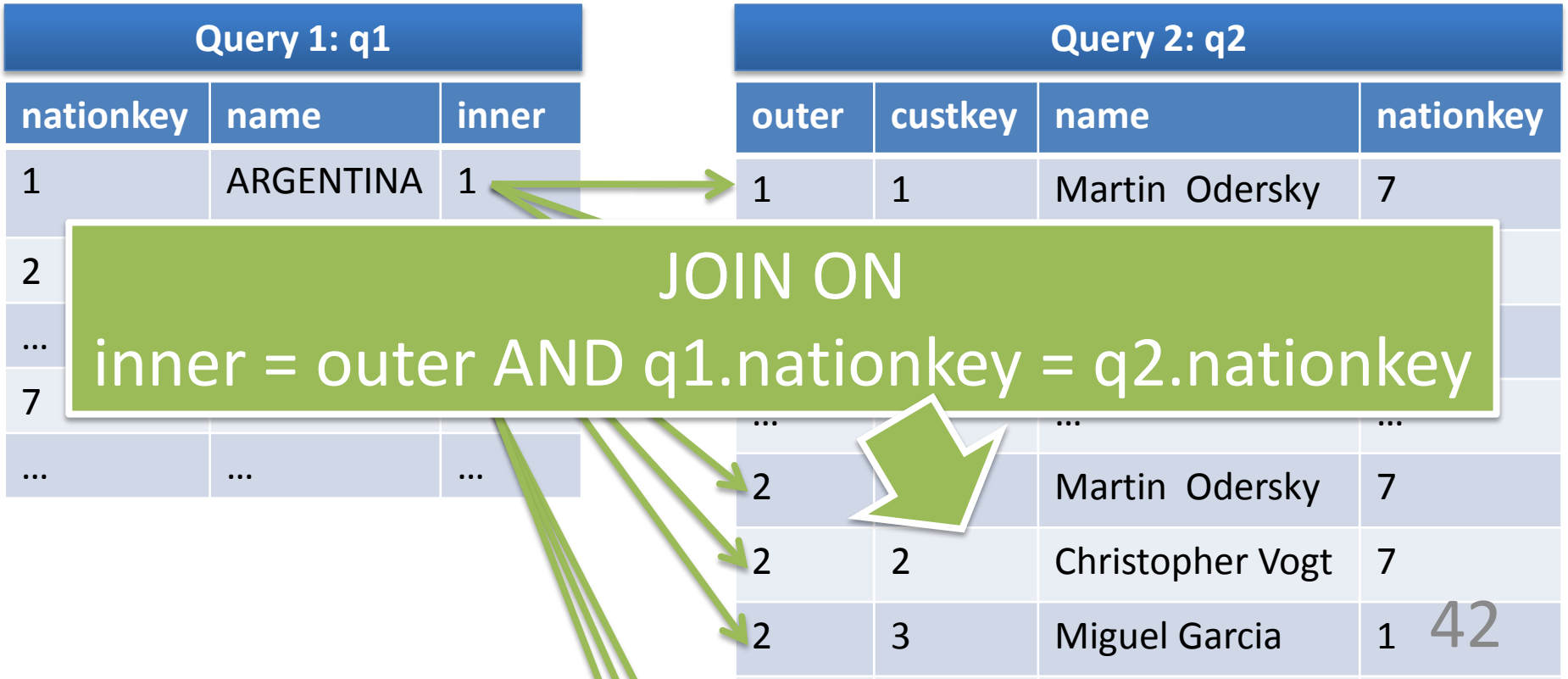
Query 2

outer	custkey	name	nationkey
1	1	Martin Odersky	7
1	2	Christopher Vogt	7
1	3	Miguel Garcia	1
…	…	…	…
2	1	Martin Odersky	7
2	2	Christopher Vogt	7
2	3	Miguel Garcia	1

Step 2: Filter all at once

customers grouped by nation

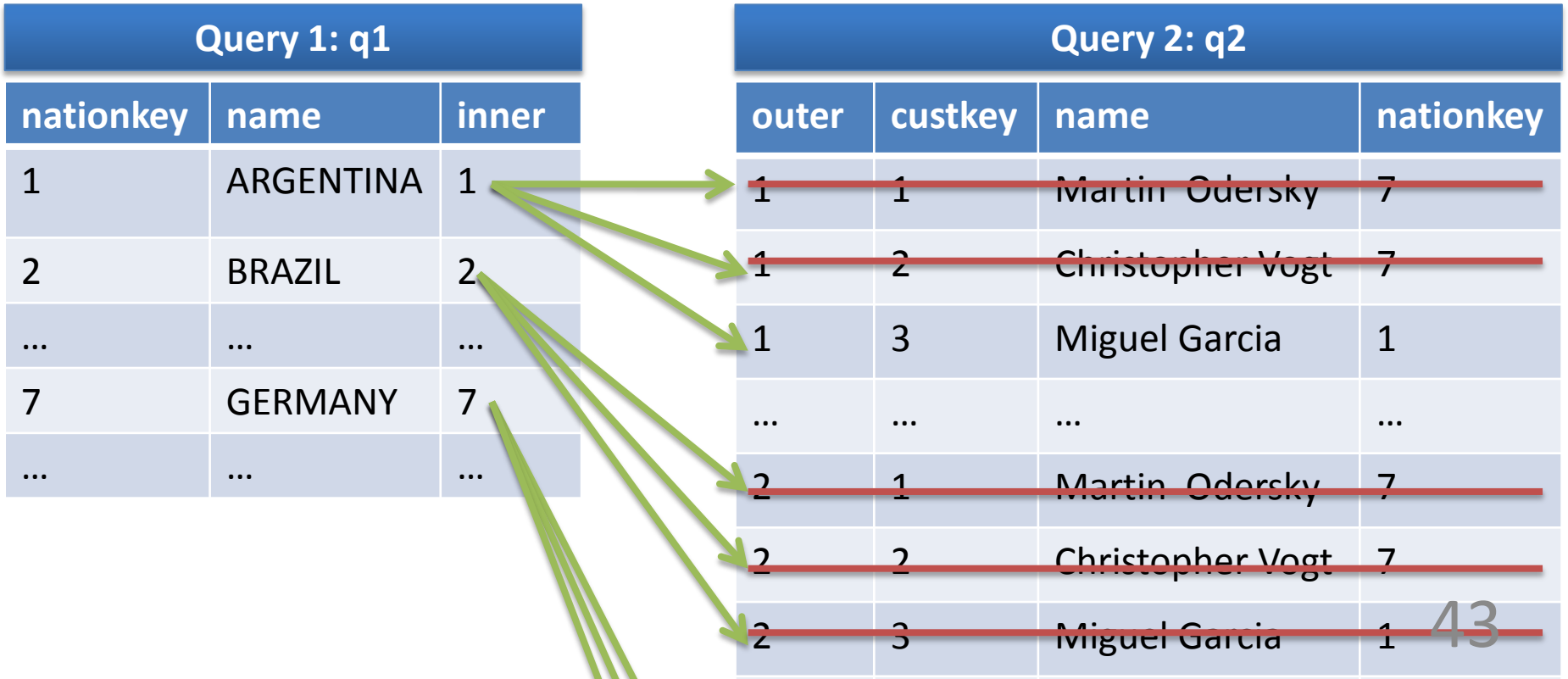
```
for( n <- nation ) yield
    customer.withFilter( _.nationkey == n.nationkey )
```



Step 2: Filter all at once

customers grouped by nation

```
for( n <- nation ) yield
    customer.withFilter( _.nationkey == n.nationkey )
```



Query 1: q1

nationkey	name	inner
1	ARGENTINA	1
2	BRAZIL	2
…	…	…
7	GERMANY	7
…	…	…

Query 2: q2

outer	custkey	name	nationkey
1	1	Martin Odersky	7
1	2	Christopher Vogt	7
1	3	Miguel Garcia	1
…	…	…	…
2	1	Martin Odersky	7
2	2	Christopher Vogt	7
2	3	Miguel Garcia	1

Nested Results

customers grouped by nation ✔

```
for( n <- nation ) yield
    customer.withFilter( _.nationkey == n.nationkey )
```

Query 1

nationkey	name	inner
1	ARGENTINA	1
2	BRAZIL	2
…	…	…
7	GERMANY	7
…	…	…

Query 2‘

outer	custkey	name	nationkey
1	3	Miguel Garcia	1
…	…	…	…
7	1	Martin Odersky	7
7	2	Christopher Vogt	7
…	…	…	…

Scala Iterables

Correlate main memory data and database tables

```
for( i <- List(17,3,4).todb; n <- nation;
                                  if i == n.nationkey ) yield n
```

One query, no avalanche:

```
WITH  t( i ) AS (VALUES (17),(3),(4))
SELECT nation.* FROM t, …
```

works for any iterable

List Order

Return items in order

```
for( i <- List(17,3,4).todb; n <- nation;
                                    if i == n.nationkey ) yield n
```

Results are in order of the in-memory list:

List(Nation(17,PERU), Nation(3,CANADA), …)

Mapping of List Order

- Get nations in order given ids
- for(i <- List(17,3,4).todb; n <- nation;
  if i == n.nationkey) yield n