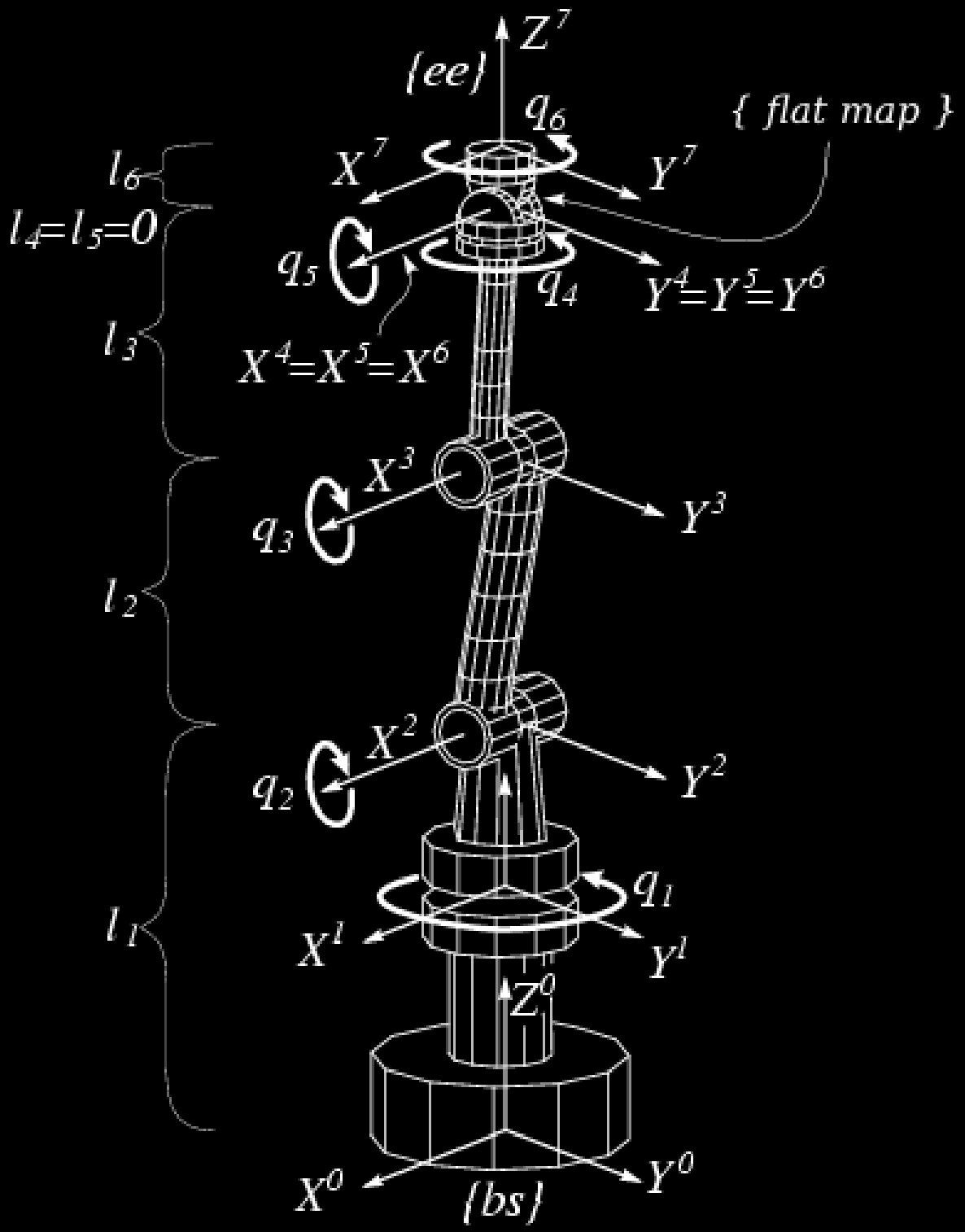

Z^7
{ee}
q_6
{ flat map }
l_6
X^7
Y^7
$l_4=l_5=0$
q_5
q_4
$Y^4=Y^5=Y^6$
l_3
$X^4=X^5=X^6$
X^3
q_3
Y^3
l_2
X^2
q_2
Y^2
q_1
l_1
X^1
Y^1
Z^0
X^0
Y^0
{bs}

Stephen Roller @stephenroller

You know what we probably need? Another Scala DSL abstracting over Hadoop.

Retweeted by Avi Bryant

5 RETWEETS	1 FAVORITE	

7:07 PM - 11 Apr 12 via Twitter for Mac · Details

Reply Retweet Favorite

* Revolute

Alex Boisvert

Work @

SQL-like query language

(for Big Data)

(Scala)
embedded
DSL

Inspired by

* Apache Hive
* Scala Query
* Cascalog

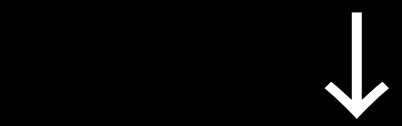
Familiar +
Type-safe +
Expressive +
Interactive

“Thin Layer”
on top of
Cascading.

A Taste of Revolute →

```
object Persons
  extends Table[(String, Int, String)]
{
  def name   = column[String]("name")
  def age    = column[Int]("age")
  def gender = column[String]("gender")
  def *      = name ~ age ~ gender
}
```

```
-- SQL
select p.* from Persons
```



```
/* Revolute */
for { p ← Persons } yield p.*
```

```
/* output multiple fields */

for { p ← Persons }
  yield p.name ~ p.age
```

```
/* filtering */

for {
  p ← Persons if (p.age > 21)
} yield p.name ~ p.age
```

```
/* combinators for complex expressions */

for {
  p ← Persons
  if (p.age > 21) && (gender === “m”)
} yield p.name ~ p.age
```

```
/* How much time are people spending per location? */

for {
  Join(p, l) ← (Persons innerJoin Locations)
                  on (_.name is _.name)
  _          ← Query.groupBy (p.name ~ l.city)
  time       ← TimeSpent(l.timestamp)
} yield p.name ~ l.city ~ time
```

```
// context provides table bindings for taps/sinks
//
// e.g.    Traffic table → HDFS or S3
//         Parter table  → MySQL
//         Summary table → Google Spreadsheet

flow(context) {

  val myQuery = for { … } yield …

  insert {
    for {
      (partner, segment, views) <- myQuery
      if segment in Set("a", "b", "c")
    } yield partner ~ views
  } into Summary
}
```

* Query
  - one or more table joined together (“join”)
  - field selection and function application (“select”)
  - one or more filters (“where”)
  - grouping and sorting (“group by”, “sort by”)
  - aggregation based on groupings (“count()”, …)

* Nest & chain queries

* 1-1, 1-0/1, 1-N mappings

* Null, Option & PartialFunction filtering

… and (eventually) more awesome.

aboisvert / **revolute**
@ github