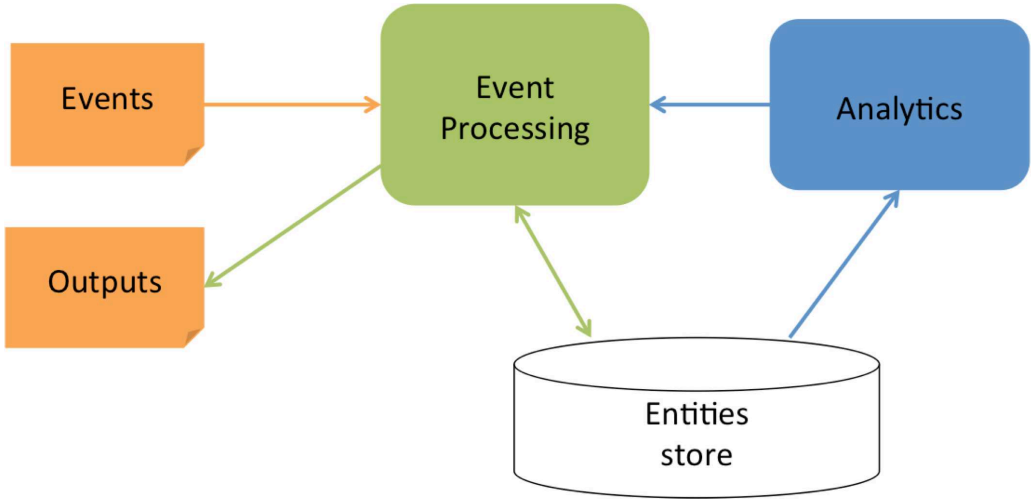
Experience Report: Prototyping a Query Compiler using Coq

Joshua Auerbach, Martin Hirzel, Louis Mandel, Avi Shinnar and Jérôme Siméon

IBM T.J. Watson Research Center

ICFP, September 2017

ODM Insights



Operational Decision Manager Insights:

- make sense of the events generated from customer interactions
- act smartly

Kinds of application:

- fraud detection
- marketing campaign

Programming model

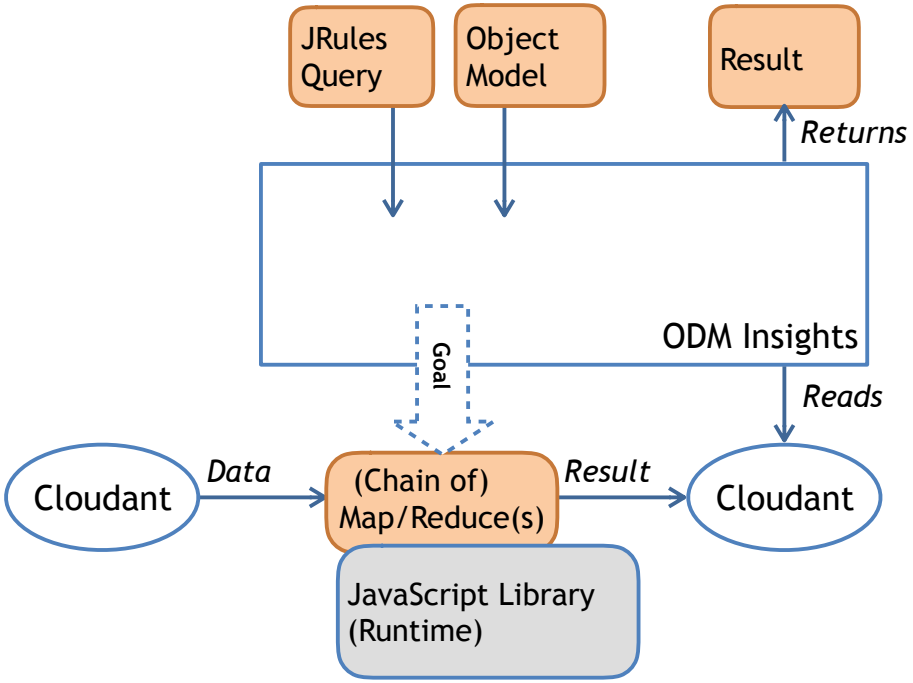
Example of marketing campaign:

```
when a purchase occurs, called LAST
if
    the total amount of all order items
     in the order items of LAST is
            more than 'average order amount'
    and the customer status of 'the customer' is NONE
then
    ...
```

Definition of a global aggregate:

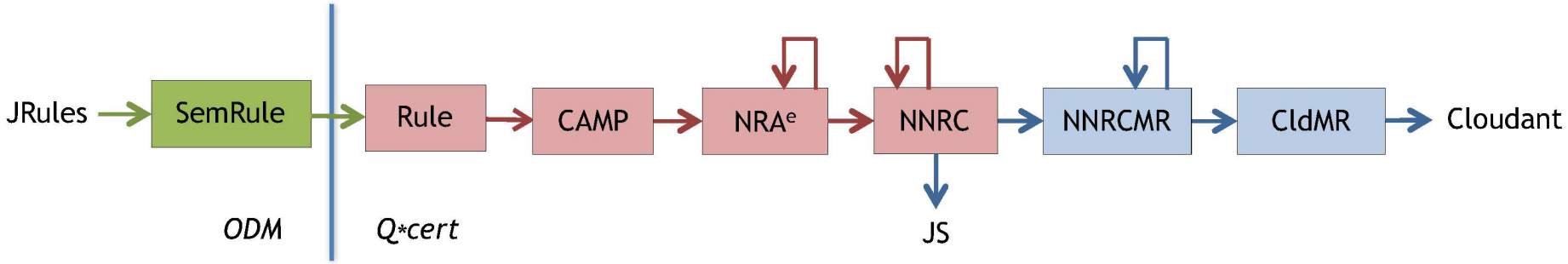
```
define 'average order amount'
  as the average amount of the order items of all orders,
  where the number of elements in the order items of each order is at least 2,
  evaluated every day at 9:28:58 AM
```

Compiler of global aggregate queries



- From JRules
  - pattern matching
  - object model
- To Cloudant
  - distributed database for JSON
  - map/reduce

Compiler architecture



From JRules to Cloudant through:

- Nested Relational Algebra with environment (NRA^e)
  - conservative extension of NRA
  - good for optimization
- Named Nested Relational Calculus Map Reduce
  - small imperative language
  - distributed model data model
  - map/reduce computations
- 6 intermediate languages, 3 optimizers

Prototype implementation in Coq

The Complete Book, Ullman

16.2.2 Laws Involving Selection

. . . . . .

To start, when the condition of a selection is complex (i.e., it involves conditions connected by AND or OR), it helps to break the condition into its constituent parts. The motivation is that one part, involving fewer attributes than the whole condition, may be moved to a convenient place where the entire condition cannot be evaluated. Thus, our first two laws for σ are the *splitting laws*:

- $\sigma_{C_1 \text{ AND } C_2}(R) = \sigma_{C_1}\big(\sigma_{C_2}(R)\big)$.
- $\sigma_{C_1 \text{ OR } C_2}(R) = \big(\sigma_{C_1}(R)\big) \cup_S \big(\sigma_{C_2}(R)\big)$.

Why Coq?:

- verify the optimizations (some are non standard)
- large semantics gap between the source and target language
- the query languages are of reasonable size
- interesting research project

Implementation: ASTs and Translations

Coq is a functional programming language:

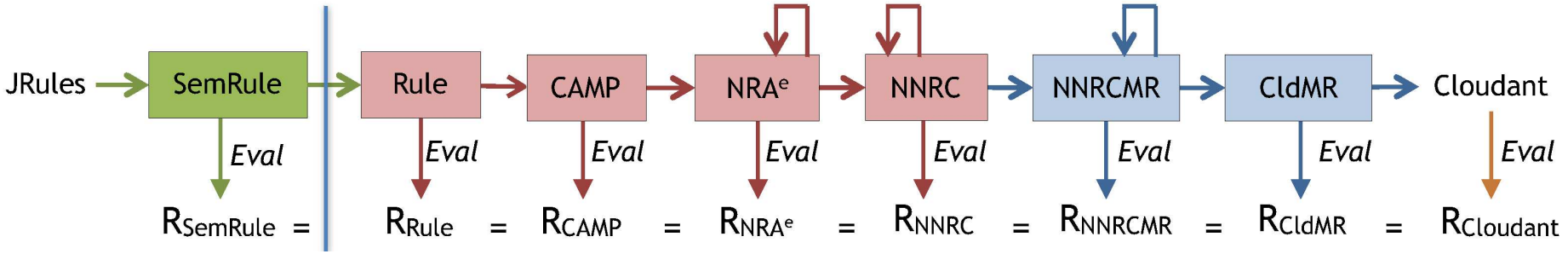
- Abstract Syntax Trees:

```
Inductive nraenv_core : Set :=
| ANID : nraenv_core
| ANConst : data -> nraenv_core
| ANMap : nraenv_core -> nraenv_core -> nraenv_core
...
```

- Translation and optimization functions:

```
Definition select_union_distr_fun q :=
  match q with
  | NRAEnvSelect q0 (NRAEnvBinop AUnion q1 q2) =>
      NRAEnvBinop AUnion (NRAEnvSelect q0 q1) (NRAEnvSelect q0 q2)
  | _ => q
  end.
```

Implementation: evaluation functions



Each intermediate language has an evaluation function:

- example

```
Definition nraenv_eval c (e:nraenv) (env:data) (x:data)
  : option data := ...
```

- it defines the semantics of the language
- it allows test execution

Implementation

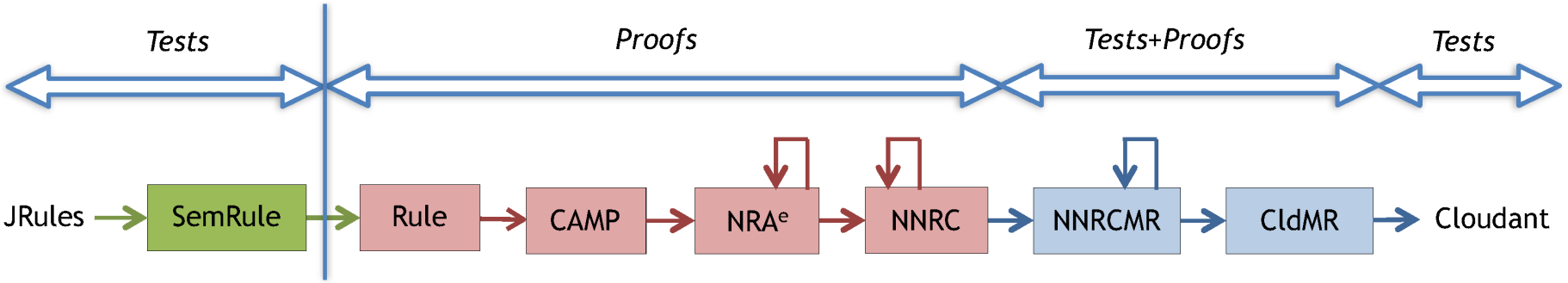
Algebraic equivalence:

```
Lemma select_union_distr q0 q1 q2 :
  σ⟨ q0 ⟩(q1 ∪ q2) ≡ σ⟨ q0 ⟩(q1) ∪ σ⟨ q0 ⟩(q2).
Proof. ... Qed.
```

Correctness proof:

```
Lemma select_union_distr_fun_correctness q:
  select_union_distr_fun q ≡ q.
Proof.
  Hint Rewrite select_union_distr : envmap_eqs.
  prove_correctness q.
Qed.
```

Choice between proof and test

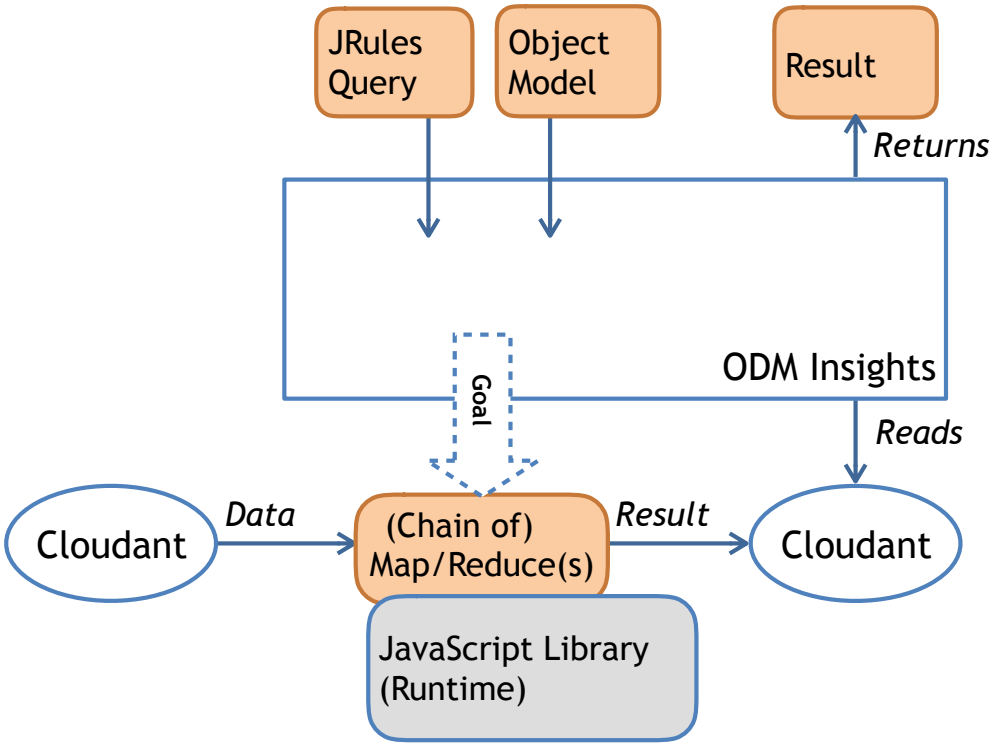


- JRules & Cloudant: semantics defined by the implementation
- Core compiler: proof of semantics preservation (including the optimizers)
- Map/Reduce compiler: test and proof of some properties:
  - well formed map/reduce chain (DAG)
  - correctness of individual rewritings

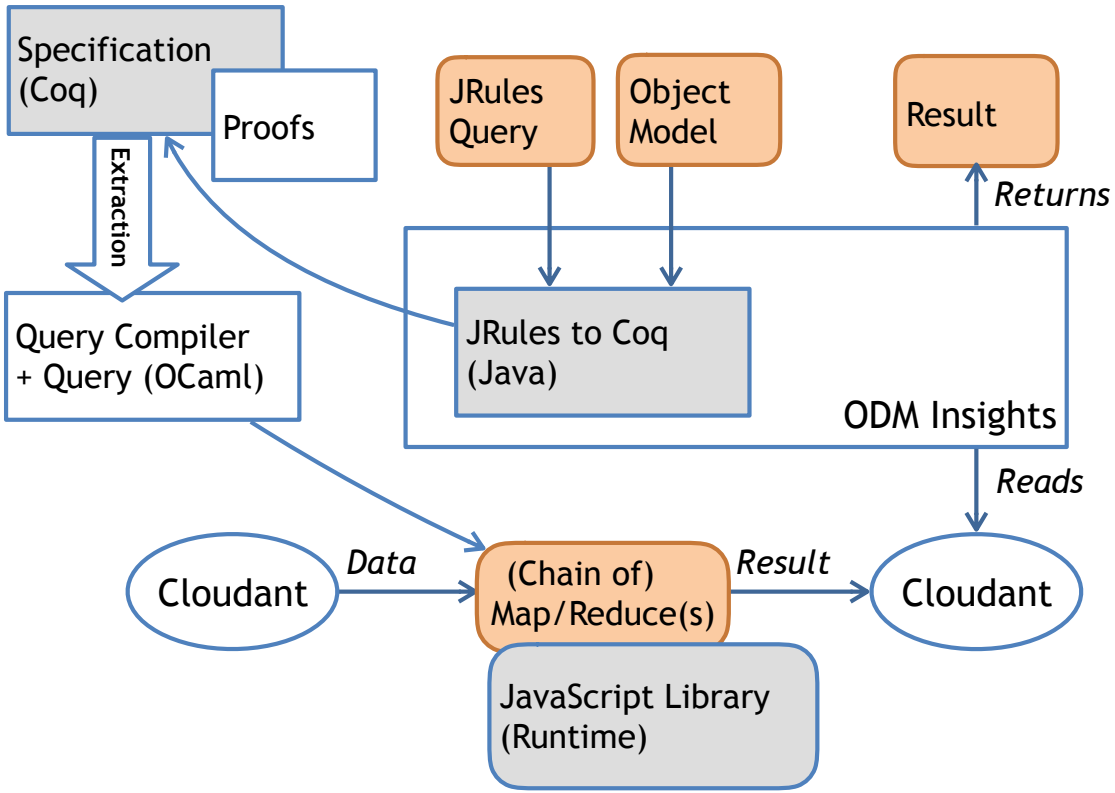
Remarks:

- Core compiler based on existing languages (and already relatively well formalized)
- Compilation of map/reduce from scratch: proofs made the experiments slower

Integration into ODM



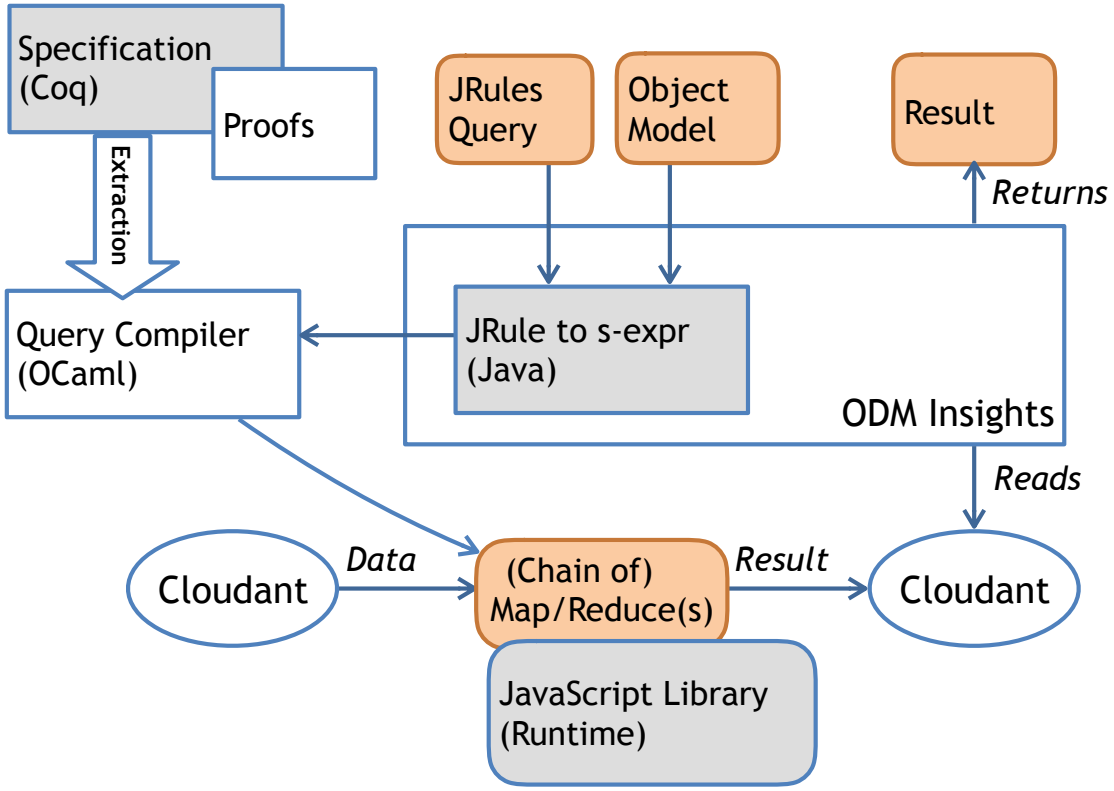
Integration into ODM



Generate Coq code from ODM:

- enables the evaluation of queries directly in Coq
- eases unitary testing
- particularly useful at the beginning of the project

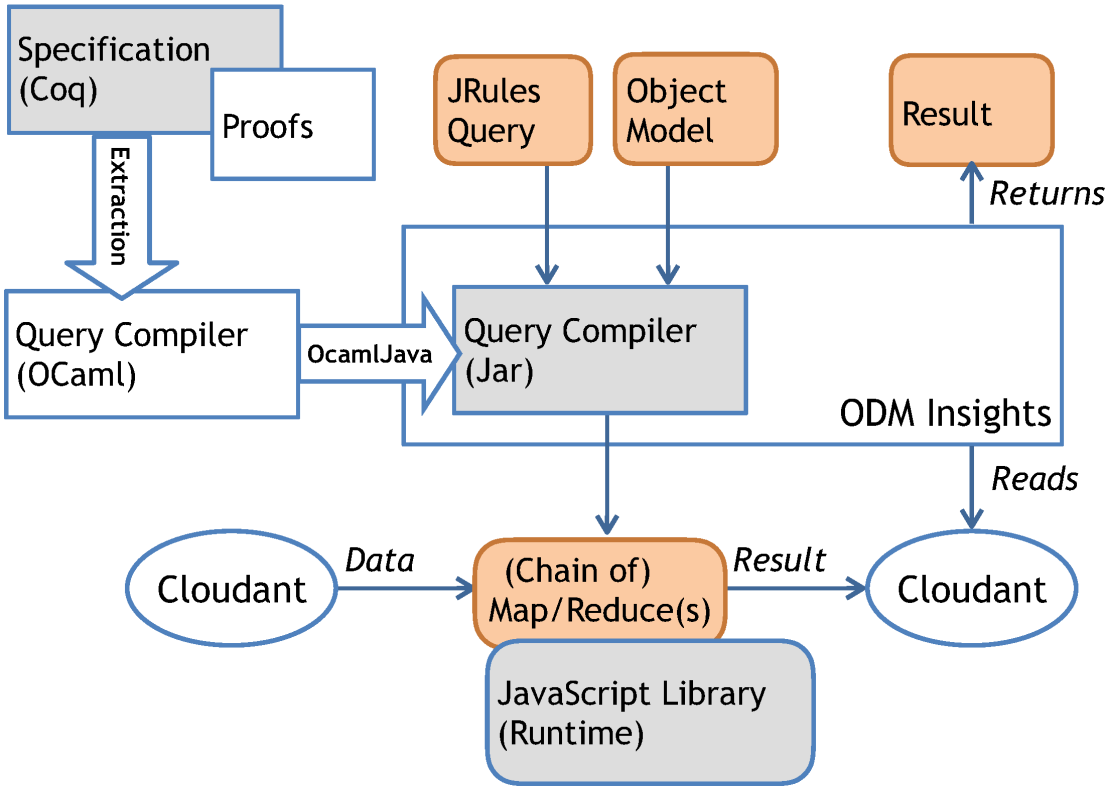
Integration into ODM



Communication between Coq and Java through s-expressions:

- provide an independent compiler
- useful for automated tests

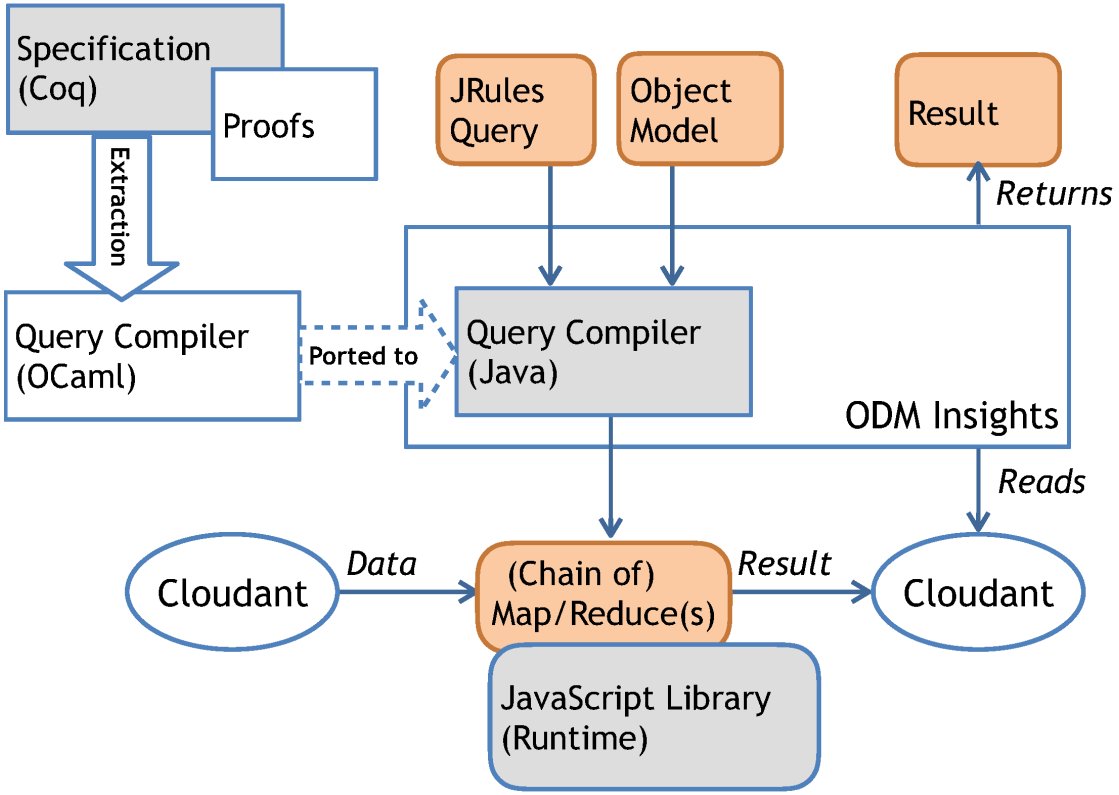
Integration into ODM



Compile the Coq compiler to a Jar file:

- extraction to OCaml and compilation using OCamlJava
- allows the integration of the Coq code in the product

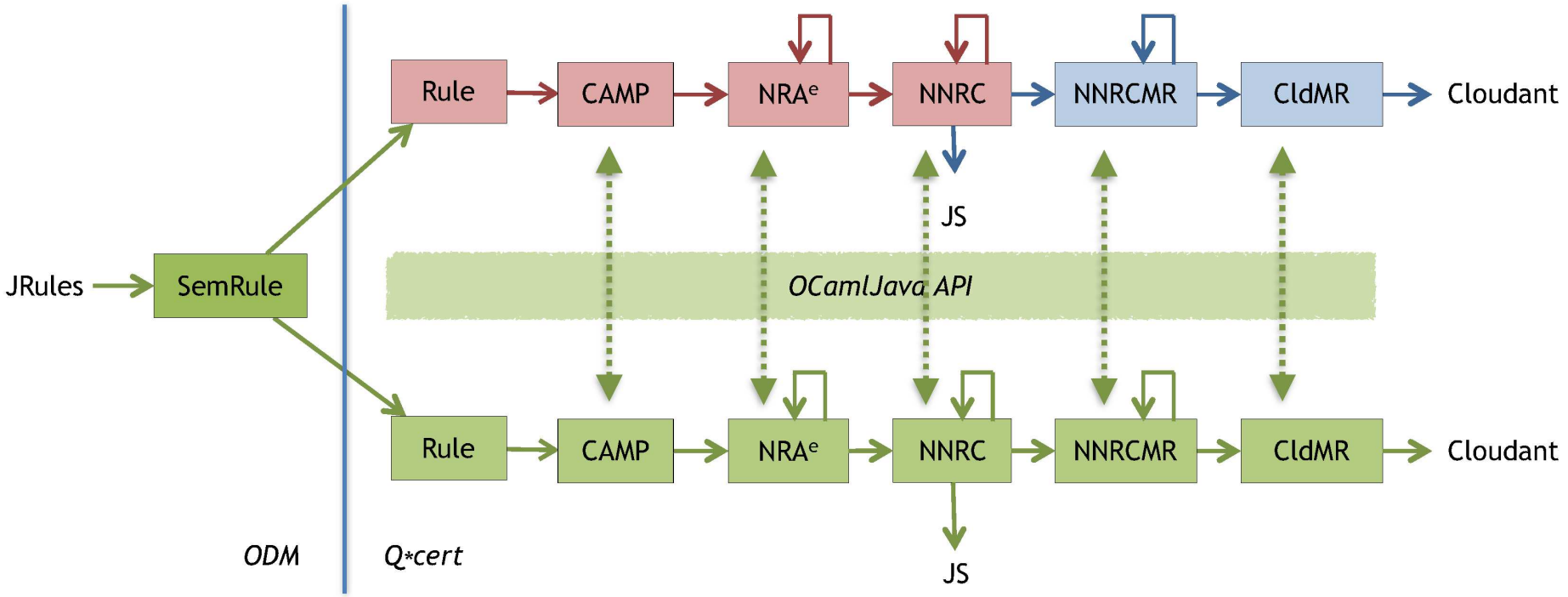
Integration into ODM



Rewrite the compiler in Java:

- goal: transfert to the product team
- idiomatic Java code
- avoid (or identify) divergences between the Coq and Java codes
- ensure that the two implementation have a similar behavior

Double compilation chain Coq-Java

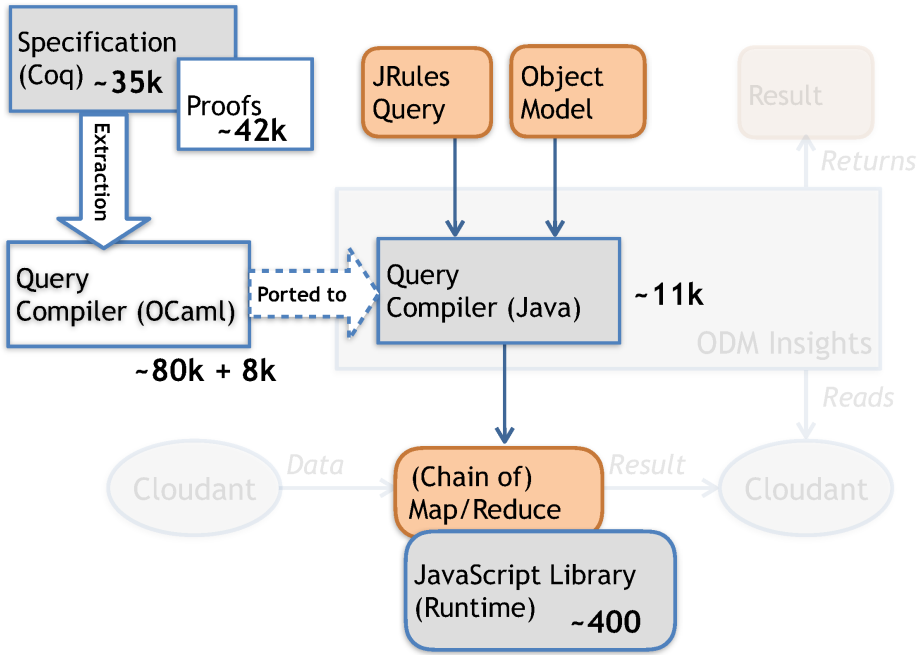


- Coq extracted to OCaml then to a jar with OCamlJava
- double compilation chain allows an arbitrary path through Coq and Java
- tests of translations = comparison of ASTs
- tests of optimizations = comparisons between traces
- translation from Coq to Java about 3–4 weeks

Java Code

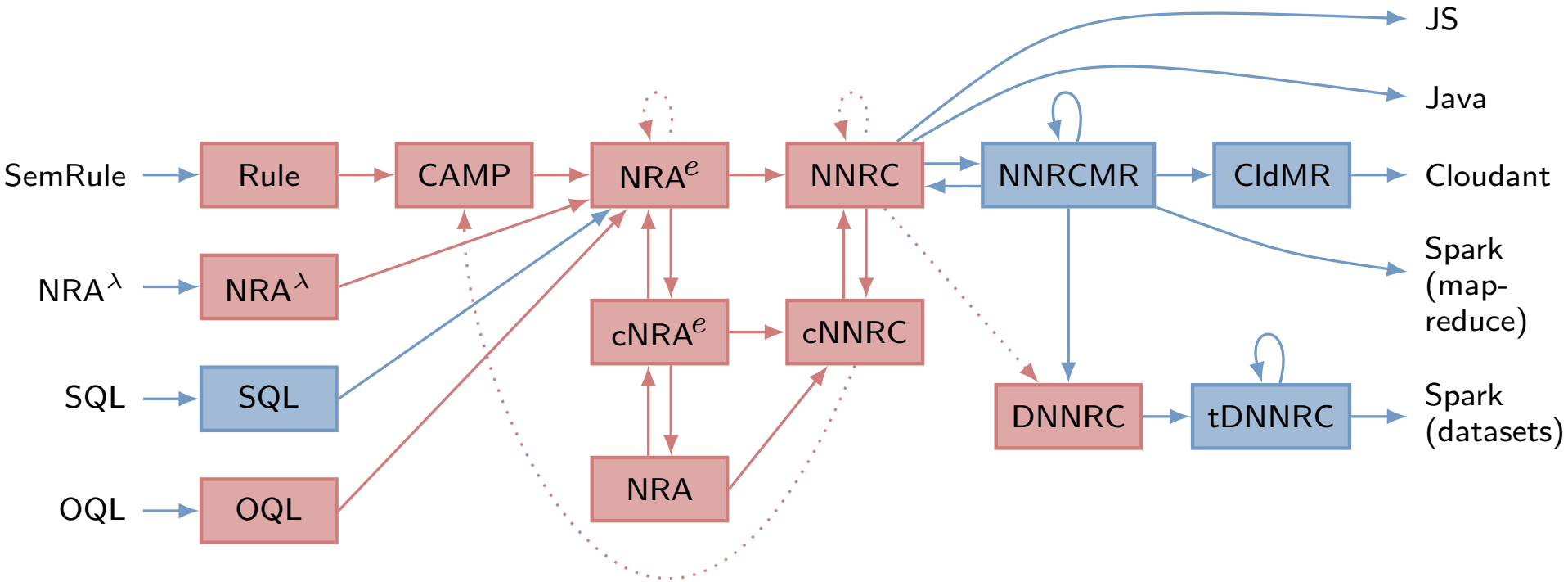
```
/** From TOptimEnvFunc.v: last checked 5/2/2016
     Definition tselect_and_fun {fruntime:foreign_runtime} (p: algenv)
      := match p with
           ANSelect op1 (ANSelect op2 op) =>
           ANSelect (ANBinop AAnd op2 op1) op
         | _ => p end. */
private static class tselect_and_fun implements OptFun {
  public NraNode optimize(NraNode nra) {
    if (nra instanceof NraSelect) {
      NraNode op1 = nra.getOperand1();
      NraNode select = nra.getOperand2();
      if (select instanceof NraSelect) {
        NraNode op2 = select.getOperand1();
        NraNode op = select.getOperand2();
        return new NraSelect(
         new NraBinaryOperator(BinaryOperator.And, op2, op1), op);
      } }
    return nra; } }
```

Some numbers



- 2014: Semantics of JRules and translation into a database algebra (7k spec, 10k proofs)
- 2015: Full compiler (optimizer, map/reduce model, code generation)
- 2016: Integration in ODM (translation to Java, tests) + open-sourcing of the research compiler
- Total: about 4 year-person

Q*cert



https://querycert.github.io

Non trivial parts or novels

Object type system for the data language [Wadlerfest'2016]

- branded values/types
- type inference
- proofs

Handling of the environment/variables in the intermediate languages [SIGMOD'2017]

- NRA^e based on combinators
- proof of equivalence with NRA
- proof that NRA optimizations still apply
- proof about scoping in NNRC

Model to introduce distribution

Compiler driver to handle compilation paths

Conclusion

Coq for prototyping

- allows a nice mix of proof and programming
- the extra development cost can be justified for certain kind of applications
- for large project having a certified part can greatly reduce debogging cost
- good surprise: adding SQL to the compiler took only 6 weeks

Current activities

- support of SQL++

Other perspectives

From prototyping to certification (Candidate: SQL to JavaScript)

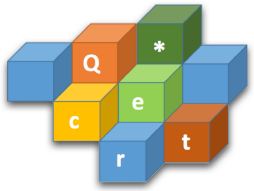
- what's SQL Semantics? what's JavaScript semantics?
- certified JavaScript runtime

Growing the query optimizer

- join reordering, query containment, query decorrelation, etc
- cost model and search space
- compilation time
- consumption by database community

More on: distribution, data updates, view maintenance

Applications: BlockChain, Node.js library, Languague-Integrated Queries



https://querycert.github.io