Experience Report: Prototyping a Query Compiler using Coq

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

IBM T.J. Watson Research Center

ICFP, September 2017

ODM Insights

ODM Insights:

- make sense of the events generated from customer interactions
- act smartly and in real-time

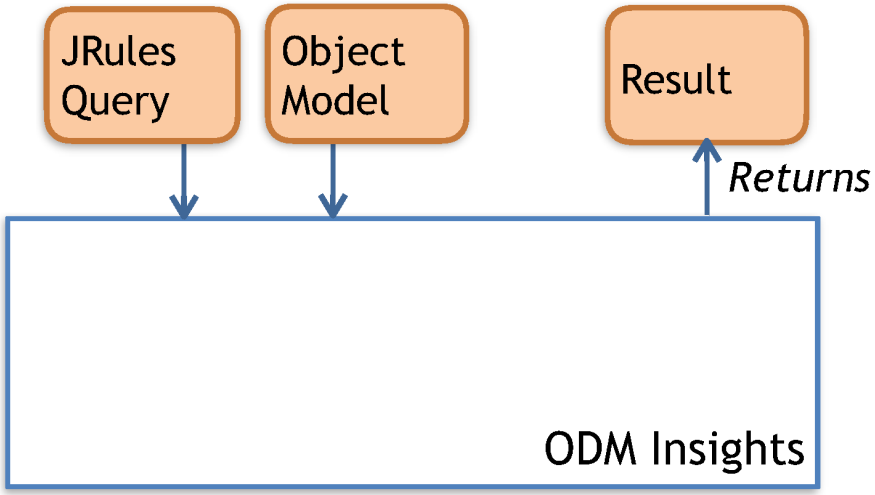
Kinds of application:

- fraud detection
- marketing campaign
- alert platform

Challenges:

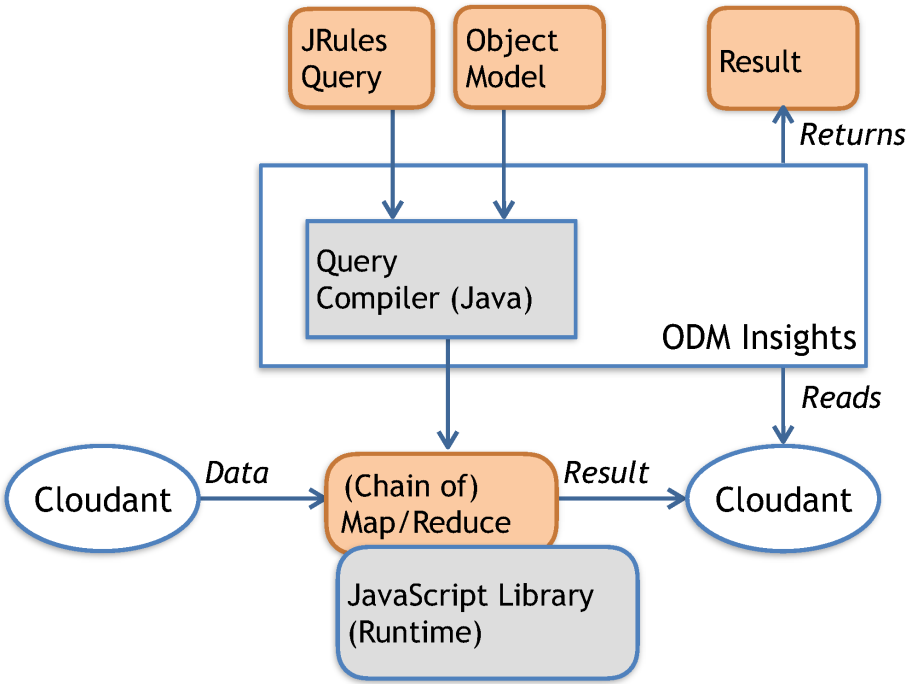
- expressive language readable by business people
- mixing event processing with analytics
- scalability

ODM Insights: a query




```
define 'test05' as detailed below,
  evaluated every minute.
definitions
  set 'test05' to the number of Customers,
       where the age of each Customer equals 32;
use 'test05' as the result.
```

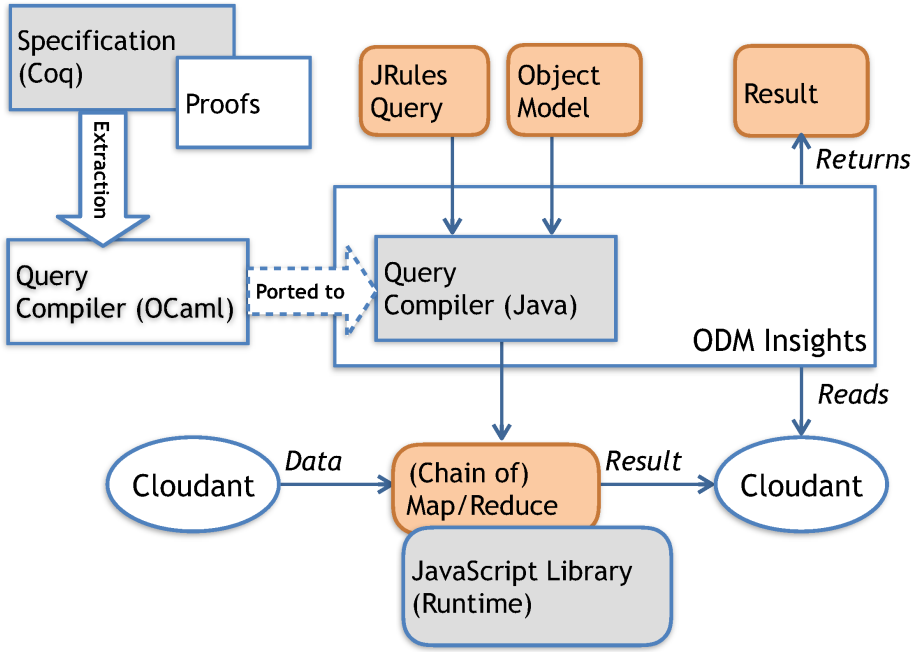
ODM Insights




```
{ "designs": [
    { "dbname": "test05b",
      "design": { "views": { "test05b": {
        "map": "function (doc) {\n  ... emit(0,srcout[iout]); ... }",
        "reduce": "_count"... } } } }
  ], ... }
```

Goal: Compiler in Java, maintained by the development team of ODM

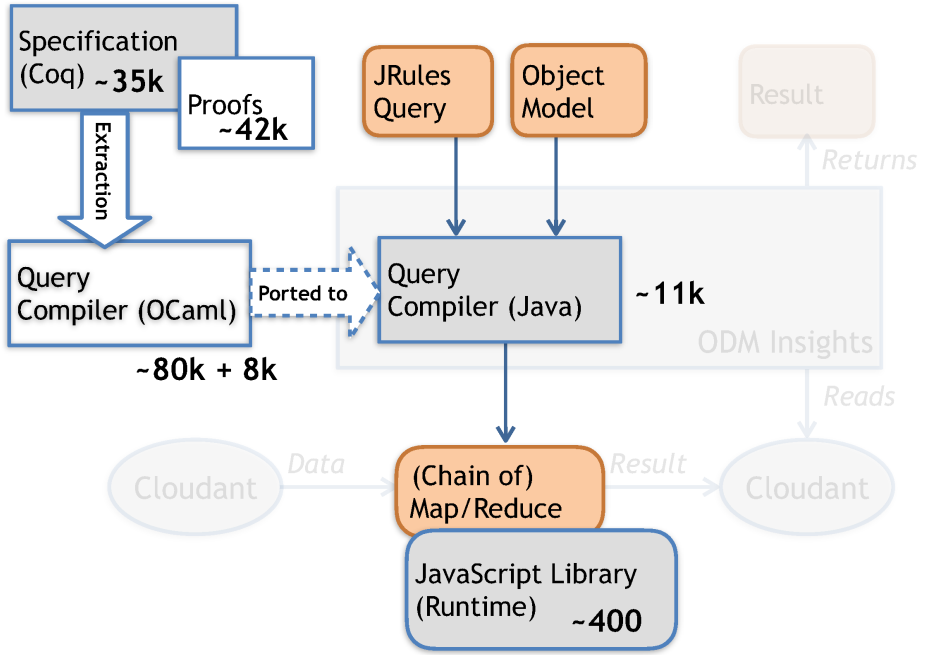
Prototyping in Coq



Why Coq?

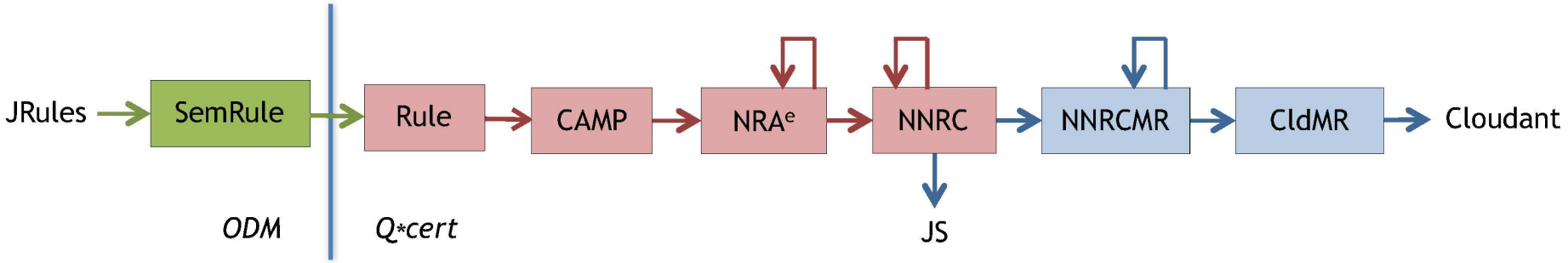
- Functional language
- Large distance between the source and target language
- Verify the optimizations (some been non standard)
- Interesting research project

Some numbers



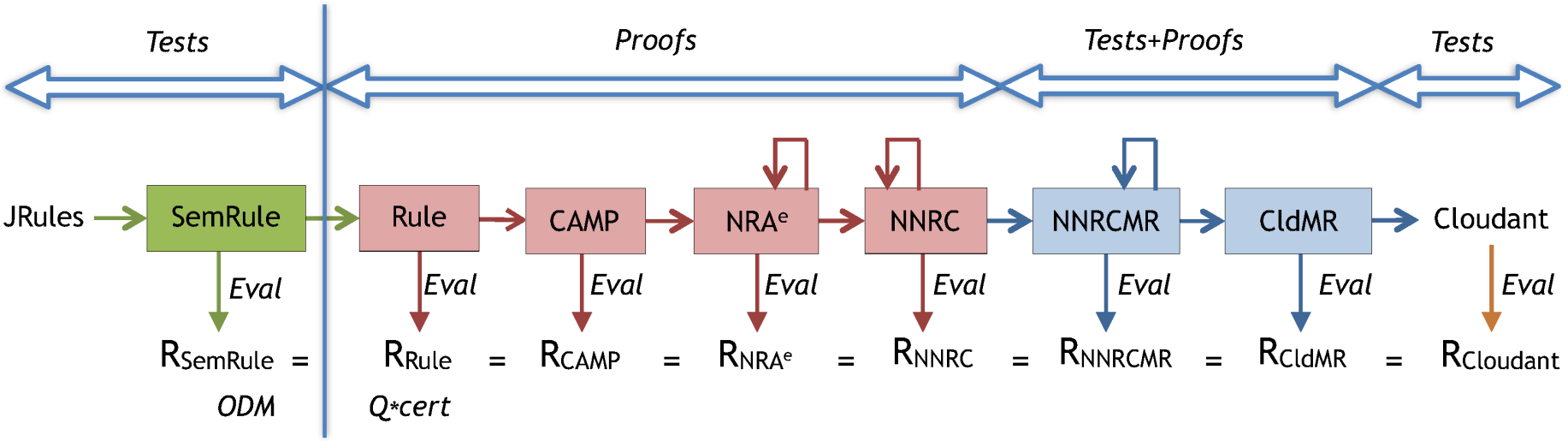
- 2014: Semantics of JRules and translation into a database algebra (7k spec, 10k proofs)
- 2015: Full compiler (optimizer, map/reduce model, code generation)
- 2016: Integration in ODM (translation to Java, tests) + open-sourcing of the research compiler
- Total: about 4 year-person

Query compiler



- From JRules (pattern matching, object model)
- To Cloudant (distributed database for JSON, map/reduce)
- Through Nested Relational Algebra (NRA^e) for the optimization
- 6 intermediate languages, 3 optimizers

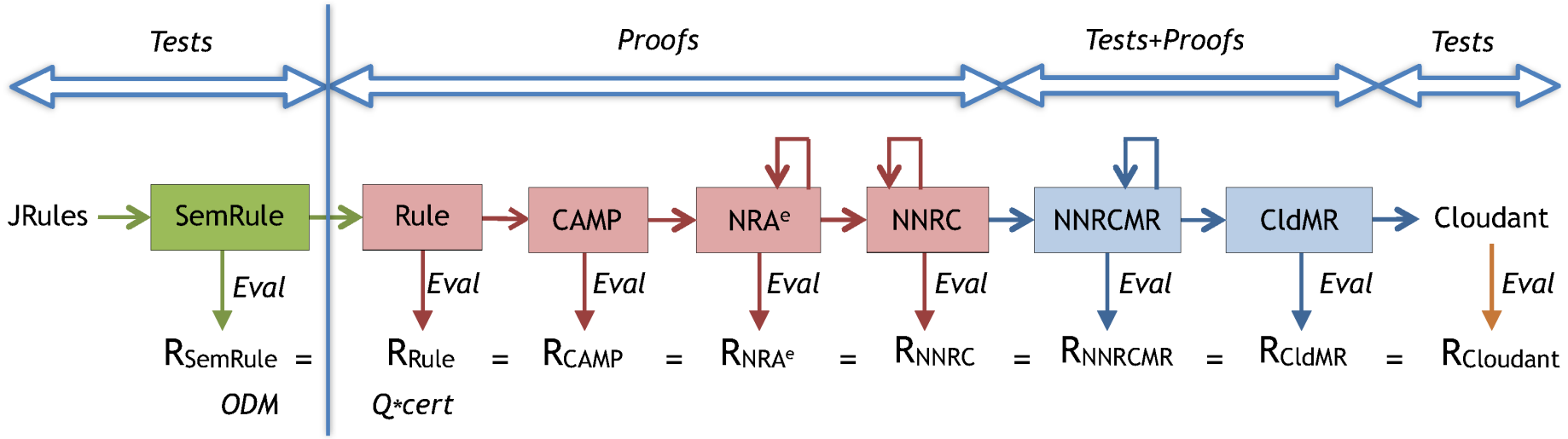
Choice between proof and test




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

- JRules & Cloudant: semantics defined by implantation
- 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

Choice between proof and test



Remarks:

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

Database Systems: The Complete Book, Ullman

Database textbook

[Database Systems: The Complete Book. Ullman et al. Second Edition]:

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*:

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

Implementation

Algebraic equivalence:

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

Functional rewrite:

```
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

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.
```

Remarks:

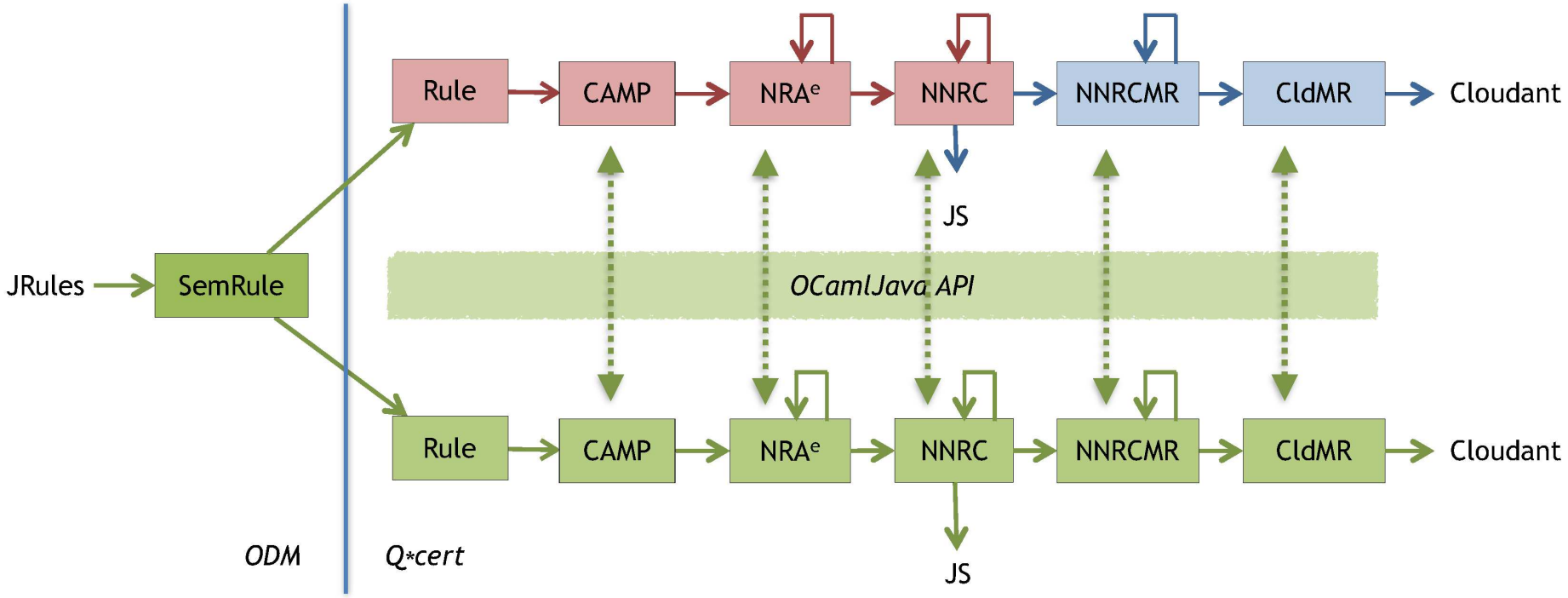
- About 100 rewrites in the optimizer
- Complex optimizations difficult to prove without formal tools
- Formal tools help to find quickly bugs

Portage vers Java

Objectifs :

- Code java 'idiomatique' qui peut etre compris et maintenu par l'équipe produit
- Éviter (ou identifier) les divergences entre le code Coq et Java
- Assurer que le comportement des deux implantations est similaire

Portage : Double chaine Coq-Java



- Coq extrait vers OCaml puis jar avec OCamlJava
- Double hélice permet une combinaison arbitraire Coq/Java
- Tests traductions = comparaisons entre ASTs
- Tests optimisations = comparisons entre traces
- Portage environ trois/quatre semaines

Portage : 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; } }
```

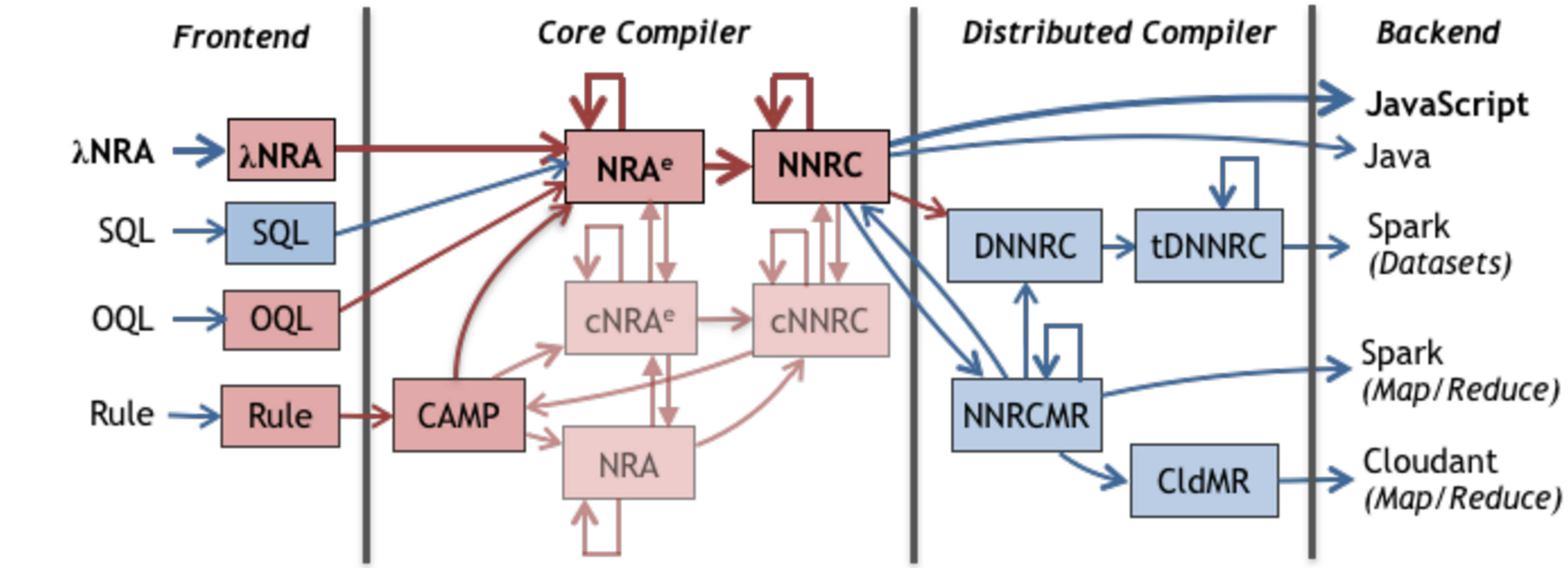
Coq : Souhaits

- Type Classes (pour les types externes) : complexité d'identifier le contexte nécéssaire dans une module (et messages d'erreur)
- Notations : source de frustration dans l'équipe (et copier coller vers LaTeX)
- Refactoring : renommage de modules, de types, de fonctions (à travers les notations, preuves, etc)
- Deboguer avec autre chose que les preuves ou `Eval`
- `Qed` vs `Defined` : Identifier les parties du code qui sont opaques

Parties non triviales ou innovantes

- Système de type à objets pour langages sur les données (méthode : branded values/types), typage et inférence de type, preuves correspondantes [Wadlerfest'2016]
- Gestion de l'environnement/variables dans les langages intermédiaires. NRA^e à base de combinateurs et preuve que les équivalences dans NRA s'appliquent à NRA^e [SIGMOD'2017]
- Modèle pour introduire la distribution (map/reduce)
- `CompilerDriver` pour gérer les chemins de compilations et les options correspondantes

Q*cert



λNRA = NRA with Lambda Terms
CAMP = Calculus of Aggregating Matching Patterns
Rule = Rule Macros for CAMP

NRA = Nested Relational Algebra
NRAᵉ = NRA with Environments
cNRAᵉ = Core NRAᵉ
NNRC = Named Nested Relational Calculus
cNNRC = Core NNRC

DNNRC = Distributed NNRC
tDNNRC = Typed DNNRC
NNRCMR = NNRC with Map/Reduce
CldMR = NNRC with Cloudant Map/Reduce

https://querycert.github.io

Conclusion

- Coq pour le prototypage (plutôt que pour certifier)
- Coût de développement supplémentaire peut être justifié dans certains cas
- Pour de gros projets, avoir une partie du code vérifiée peut réduire de façon importante les coûts de debogue
- Bonne surprise : ajouter SQL au compilateur en 6 semaines (grâce à des langages intermédiaires bien défini)
- Un compilateur de requêtes réellement certifié est envisageable