DataCert

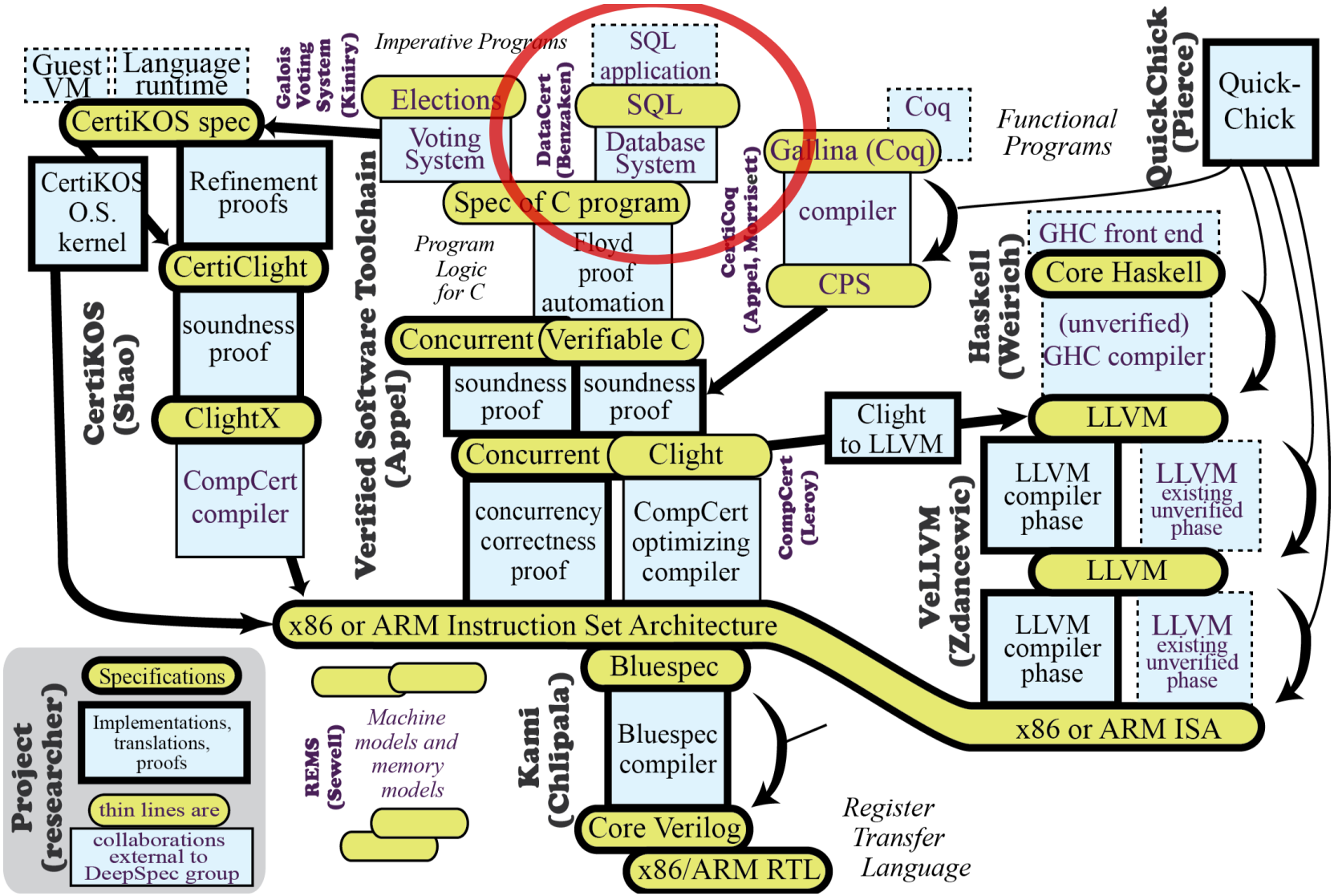
Data Centric Systems Deep Specification

V. Benzaken, É. Contejean

LRI - CNRS - Université Paris Sud
VALS Research Group
Verification of Algorithms, Languages and Systems

DeepSpec workshop - Princeton - June 6-8 2016

DeepSpec: The Big Project



Motivations

Data are pervasive and valuable ...

Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees?

Motivations

Data are pervasive and valuable ...

Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees? By using formal methods

Promising approach: use of proof assistants such as Coq or Isabelle (e.g., Compcert)

Seminal work by [Malecha et.al] POPL2010

Motivations

Data are pervasive and valuable ...

Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees?

Promising approach: use of proof assistants such as Coq or Isabelle (e.g., Compcert)

Seminal work by [Malecha et.al] POPL2010 Let's do it!

Long term goal:

Data Centric Programming Languages and System's Coq formalization

Towards Data Centric Systems Deep Specification

First systems of interest: Relational ones

. . . **relational database systems** still play a central role

Beyond relational: NoSQL engines, XML, RDF . . .

Keeping in mind relevant database applications

Education aware: A first Coq formalised course in database systems

Relational model then **SQL**

- Information modeling:

  through relations and tuples

Structure: relation name and sort (finite set of attributes)

r(a, b) relation name **r** sort: {a,b}

- Information extraction:

  through query languages (relational algebra)

Relational Model Deep Specification

Two perspectives:

named vs unnamed

$\mathbf{r} = \{\mathsf{f1}; \mathsf{f2}; \mathsf{f3}\}$

$\mathsf{f1}(\mathsf{a}) = 1, \mathsf{f1}(\mathsf{b}) = 2$
$\mathsf{f2}(\mathsf{a}) = 3, \mathsf{f2}(\mathsf{b}) = 2$
$\mathsf{f3}(\mathsf{a}) = 1, \mathsf{f3}(\mathsf{b}) = 1$

$\mathbf{r} = \{(1, 2); (3, 2); (1, 1)\}$

Relational Model Deep Specification

named (SPJR)

$$q := r \mid \sigma_f(q) \mid \pi_W(q) \mid \rho_g(q) \mid q \bowtie q \mid q \cup q \mid q \cap q \mid q \setminus q$$

unamed (SPC)

$$q := r \mid \sigma_f(q) \mid \pi_W(q) \mid q \times q \mid q \cup q \mid q \cap q \mid q \setminus q$$

Relational Model Deep Specification

[Benzaken et.al] ESOP 2014

More than 20.000 Coq loc

Based on two textbooks:

Abiteboul, Hull, Vianu: Foundations of Databases. Addison-Wesley (1995)

Ullman: Principles of Database Systems, 2nd Edition. Computer Science Press (1982)

We learned a lot

- subtleties hidden in textbooks
- relevant hypotheses
- useless concepts
- sloppy definitions
- lack of formal specifications

SQL: a Simple Declarative Language

SQL "inter-galactic" dialect for manipulating (relational) data

Declarative DSL describe what opposed as how

With attribute's names as first-class citizens

$\Rightarrow$ name-based perspective

SQL: a Simple Declarative Language

Assuming `tbl1(a,b,c)` and `tbl2(d,e)`

```
select a, c from tbl1 where b>3;
```

$$\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1}))$$

```
select a as a1, c as c1 from tbl1 where b>3;
```

$$\rho_{\{\mathtt{a}\to\mathtt{a1};\mathtt{c}\to\mathtt{c1}\}}(\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1})))$$

```
select * from tbl1, tbl2 where b=d and c=e;
```

$$\sigma_{\mathtt{b}=\mathtt{d}\wedge\mathtt{c}=\mathtt{e}}(\mathtt{tbl1}\bowtie\mathtt{tbl2})$$

SQL: a Simple Declarative Language

Assuming `tbl1(a,b,c)` and `tbl2(d,e)`

```
select a, c from tbl1 where b>3;
```

$$\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1}))$$

```
select a as a1, c as c1 from tbl1 where b>3;
```

$$\rho_{\{\mathtt{a}\to\mathtt{a1};\mathtt{c}\to\mathtt{c1}\}}(\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1})))$$

```
select * from tbl1, tbl2 where b=d and c=e;
```

$$\sigma_{\mathtt{b}=\mathtt{d}\wedge\mathtt{c}=\mathtt{e}}(\mathtt{tbl1}\bowtie\mathtt{tbl2})$$

SQL: a Simple Declarative Language

Assuming `tbl1(a,b,c)` and `tbl2(d,e)`

```
select a, c from tbl1 where b>3;
```

$$\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1}))$$

```
select a as a1, c as c1 from tbl1 where b>3;
```

$$\rho_{\{\mathtt{a}\to\mathtt{a1};\mathtt{c}\to\mathtt{c1}\}}(\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1})))$$

```
select * from tbl1, tbl2 where b=d and c=e;
```

$$\sigma_{\mathtt{b}=\mathtt{d}\wedge\mathtt{c}=\mathtt{e}}(\mathtt{tbl1} \bowtie \mathtt{tbl2})$$

SQL: a Simple Declarative Language

```
select b, 2*(a+c), sum(a)
from tbl1
where a+b = 7
group by b, a+c
having b > 6;
```

No algebraic semantics

SQL: a Simple Declarative Language

Declarative DSL = ... simple

But

Not so simple ...

SQL: a Simple Declarative Language

Based on relational algebra for the `select-from-where` part

Mixes two algebras: the name based SPJR and the unamed SPC

$\Longrightarrow$ strange behaviours

SQL: Examples

Some queries are rejected while they admit an equivalent algebraic semantics

```
select * from tbl, tbl;
```

Computes either an auto-join or a Cartesian product (up to attribute renaming)

SQL: Examples

Some queries are rejected while they admit an equivalent algebraic semantics

```
select * from tbl, tbl;
ERROR: table name "tbl" specified more than once
```

Computes either an auto-join or a Cartesian product (up to attribute renaming)

SQL: Examples

Some are accepted but should not be

```
select a as c, b as c from tbl;
```

is unduly accepted ... but

SQL: Examples

Some are accepted but should not be

```
select a as c, b as c from tbl;
```

is unduly accepted ... but

cannot be embedded as a subquery

```
select c from (select a as c, b as c from tbl) as t
ERROR: column reference "c" is ambiguous
```

SQL: Examples

Some are accepted but should not be

```
select a as c, b as c from tbl;
```

is unduly accepted ... but

cannot be embedded as a subquery

```
select c from (select a as c, b as c from tbl) as t
ERROR: column reference "c" is ambiguous
```

Reformulation

```
select a as ca, b as cb from tbl;
```

SQL: Examples

Attributes' order matters

```
table tbl1;
```

```
 a1 | b1 | c1
----+----+---
 1  | 2  | 3
 4  | 5  | 7
(2 rows)
```

```
(select b1, a1, c1 from tbl1)
          intersect
(select a1, b1, c1 from tbl1);
```

```
 b1 | a1 | c1
----+----+---
(0 rows)
```

SQL: Examples

Attributes' order matters

```
table tbl1;
```

```
 a1 | b1 | c1
----+----+---
 1  | 2  | 3
 4  | 5  | 7
(2 rows)
```

```
(select b1, a1, c1 from tbl1)
          intersect
(select a1, b1, c1 from tbl1);
```

```
 b1 | a1 | c1
----+----+---
(0 rows)
```

$\cap$ and $\cup$ are not idempotent neither are they associative nor commutative

SQL: Foundational Mismatch

Attribute names are first class citizens

$\Rightarrow$ **name based algebra**

but **unamed** algebra is used

Under database programmer responsability to **manage names in an unamed setting**

Clean name-based formal specification and mechanised semantics are missing

SQL: Foundational Mismatch

Attribute names are first class citizens

$\Rightarrow$ **name based algebra**

but **unamed** algebra is used

Under database programmer responsability to **manage names in an unamed setting**

Clean name-based formal specification and mechanised semantics are missing

$\Longrightarrow$ **SQLCert**

Formal Framework: SQLCert

Based on attributes names

Conforming to theoretical foundations

Allow to detect and manage such queries

Define SQL_{Coq}: a formal SQL-friendly grammar

Notion of well-formed queries

`select-from-where-` `group-by-having` with function symbols, aggregates and nested queries

Formal Framework: SQLCert

Define $[\![\_]\!]_{\mathrm{SQL}_{\mathrm{coq}}}$ (`eval_sql_query`) a mechanised semantics for $\mathrm{SQL}_{\mathrm{Coq}}$.

Accepts all well-formed queries

Slight bias between SQL and $\mathrm{SQL}_{\mathrm{Coq}}$ compensated thanks to a reformulation

$\mathrm{SQL}_{\mathrm{Coq}}$: Syntax

```
Inductive sql_query : Type :=
  | Sql_Table : relname → sql_query
  | Sql_Set : set_op → sql_query → sql_query → sql_query
  | Sql_Select :
      (* select *) select_item →
      (* from *) list from_item →
      (* where *) sql_formula →
      (* group by *) group_by →
      (* having *) sql_formula → sql_query
with from_item : Type := [...]
with sql_formula : Type := [...]
with sql_atom : Type :=
  | Sql_True : sql_atom
  | Sql_Pred : predicate → list aggterm → sql_atom
  | Sql_Quant :
    quantifier -> predicate -> list aggterm -> sql_query -> sql_atom
  | Sql_In :  select_item -> sql_query -> sql_atom.
```

$\mathrm{SQL}_{\mathrm{Coq}}$: Syntax

```
Inductive sql_query : Type :=
  | Sql_Table : relname → sql_query
  | Sql_Set : set_op → sql_query → sql_query → sql_query
  | Sql_Select :
      (* select *) select_item →
      (* from *) list from_item →
      (* where *) sql_formula →
      (* group by *) group_by →
      (* having *) sql_formula → sql_query
with from_item : Type := [...]
with sql_formula : Type := [...]
with sql_atom : Type :=
  | Sql_True : sql_atom
  | Sql_Pred : predicate → list aggterm → sql_atom
  | Sql_Quant :
    quantifier -> predicate -> list aggterm -> sql_query -> sql_atom
  | Sql_In :  select_item -> sql_query -> sql_atom.
```

SQL_{Coq}: Semantics

```
Fixpoint eval_sql_query (sq : sql_query) {struct sq} : setT :=
  match sq with
  | Sql_Table tbl => instance tbl
  | Sql_Set o sq1 sq2 =>
     if sql_sort sq1 =S?= sql_sort sq2 then [...] else emptySE
  | Sql_Select s lsq w gby h =>
     let elsq := (* evaluation of the from part [lsq] *)
         map eval_from_item lsq in
     let cc := (* selection of the from part by the where formula [w]
                  (with old names) *)
         filter (fun t => eval_sql_formula w (Tpl t))
                (plain_product elsq) in
     let lg1 := (* computation of the groups grouped according to [gby] *)
         make_groups cc gby in
     let lg2 := (* discarding groups according the having clause [h] *)
         filter (fun g => eval_sql_formula h (Grp g)) lg1 in
       (* applying the outermost projection and renaming,
          the select part [s] *)
       Set.map (fun g => projection s (Grp g)) lg2
  end with [...]
```

Formal Framework: SQLCert

Related to an (extended) relational algebra (`eval_equery`)

Coq mechanised parser $\mathscr{T}$ (`sql_to_alg`), of $\mathsf{SQL}_{\mathsf{Coq}}$ to our, name-based algebra with its Coq adequation proof and its Ocaml extraction

Adequation Theorem

Theorem

Let sq be a well-formed SQL_{Coq} query then

$$[\![sq]\!]_{SQL_{coq}} = [\![\mathscr{T}(sq)]\!]_{ALG_{Ext}}$$

```
Lemma sql_to_alg_is_sound :
∀ ip is ia I, well_sorted_instance I →
  ∀ n q sq, sql_to_alg n sq = Translation q →
     well_formed_sql_query sq = true →
      eval_sql_query ip is ia I sq =SE=
      eval_equery (interp_predicate ip) (interp_symbol is ia) ia I q).
```

Adequation Theorem

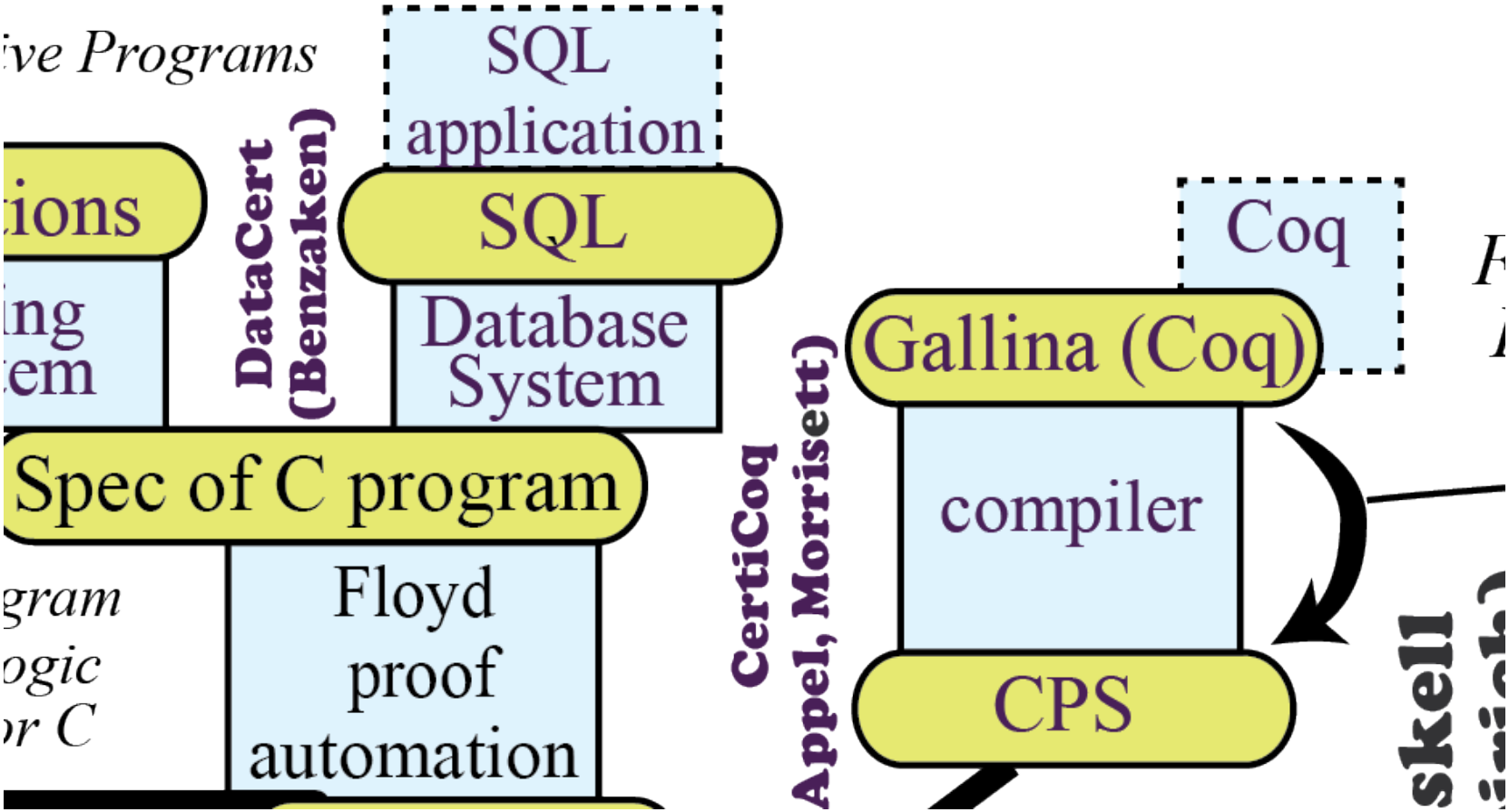
Theorem

Let sq be a well-formed SQL_{Coq} query then

$$[\![sq]\!]_{SQL_{coq}} = [\![\mathscr{T}(sq)]\!]_{ALG_{Ext}}$$

```
Lemma sql_to_alg_is_sound :
∀ ip is ia I, well_sorted_instance I →
  ∀ n q sq, sql_to_alg n sq = Translation q →
     well_formed_sql_query sq = true →
      eval_sql_query ip is ia I sq =SE=
      eval_equery (interp_predicate ip) (interp_symbol is ia) ia I q).
```

DataCert and DeepSpec

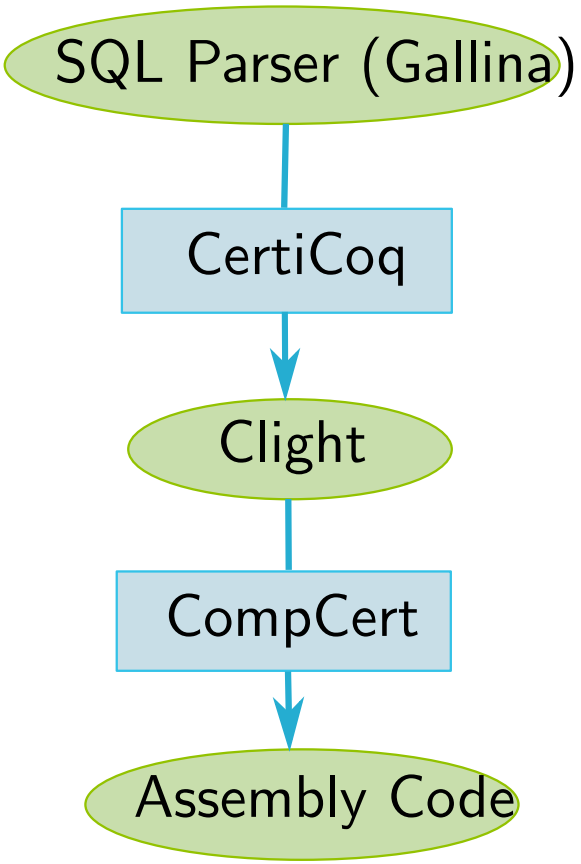


Short Term: Certified SQL Query Evaluation

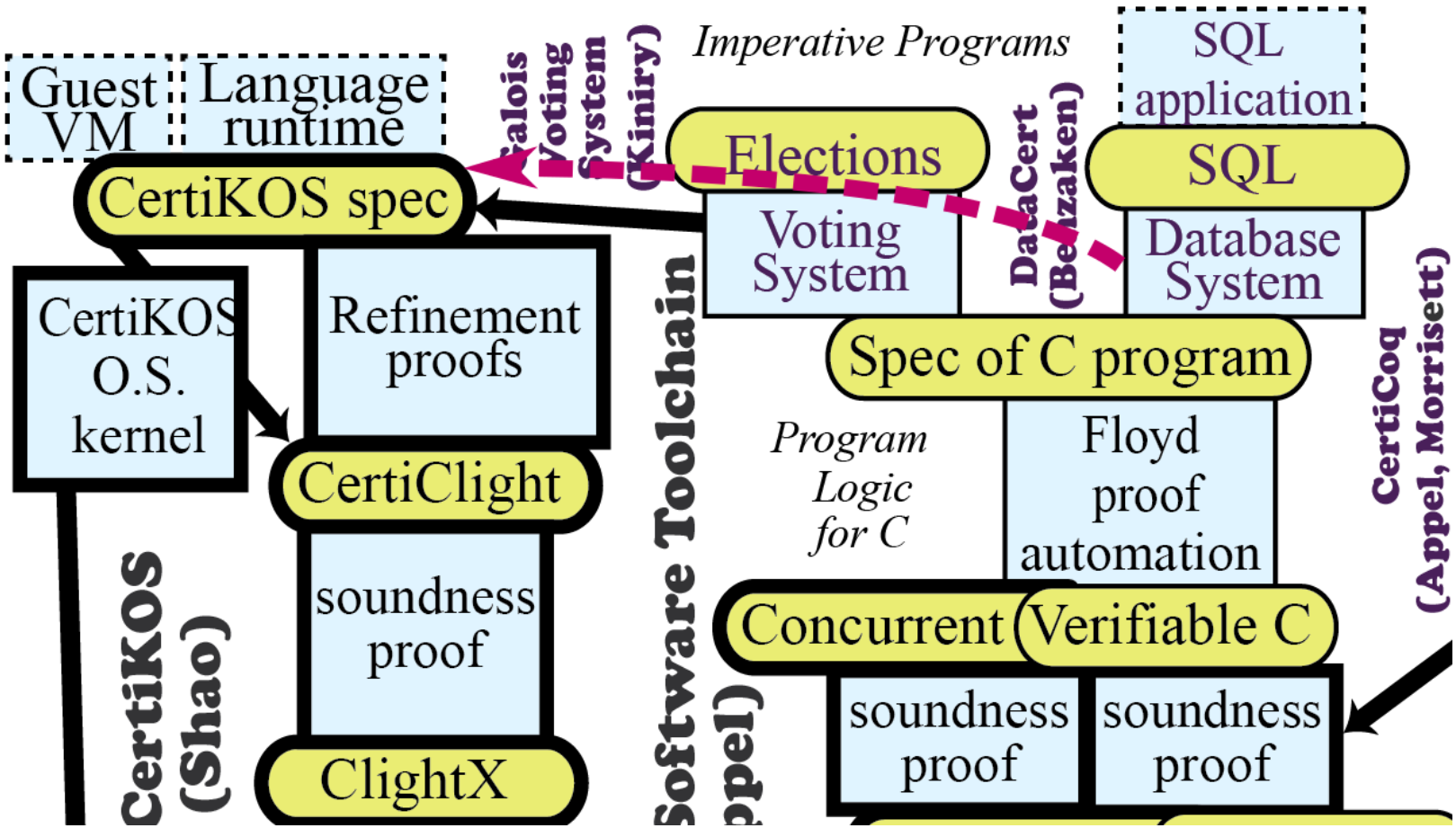
Use CertiCoq

SQL_{Coq} queries translated into Clight

Compare its execution times on various DB instances



Long Term: A Minimal Realistic Relational Engine



Long Term: A Minimal Realistic Relational Engine

Real SQL engines/relational databases' implementations are very involved and have a long-standing history

Their code is not modular at all

Formally verifying them could be a painful task

Rather: deeply specify a small, realistic database based on mCertiKOS Hypervisor.

Conclusions

SQLCert a realistic formalisation of SQL

Based on attributes names

Conforming to theoretical foundations

Around 32.000 Coq loc

Next step: query planner

Also: updates + data integrity and privacy