Language, Data, and Security

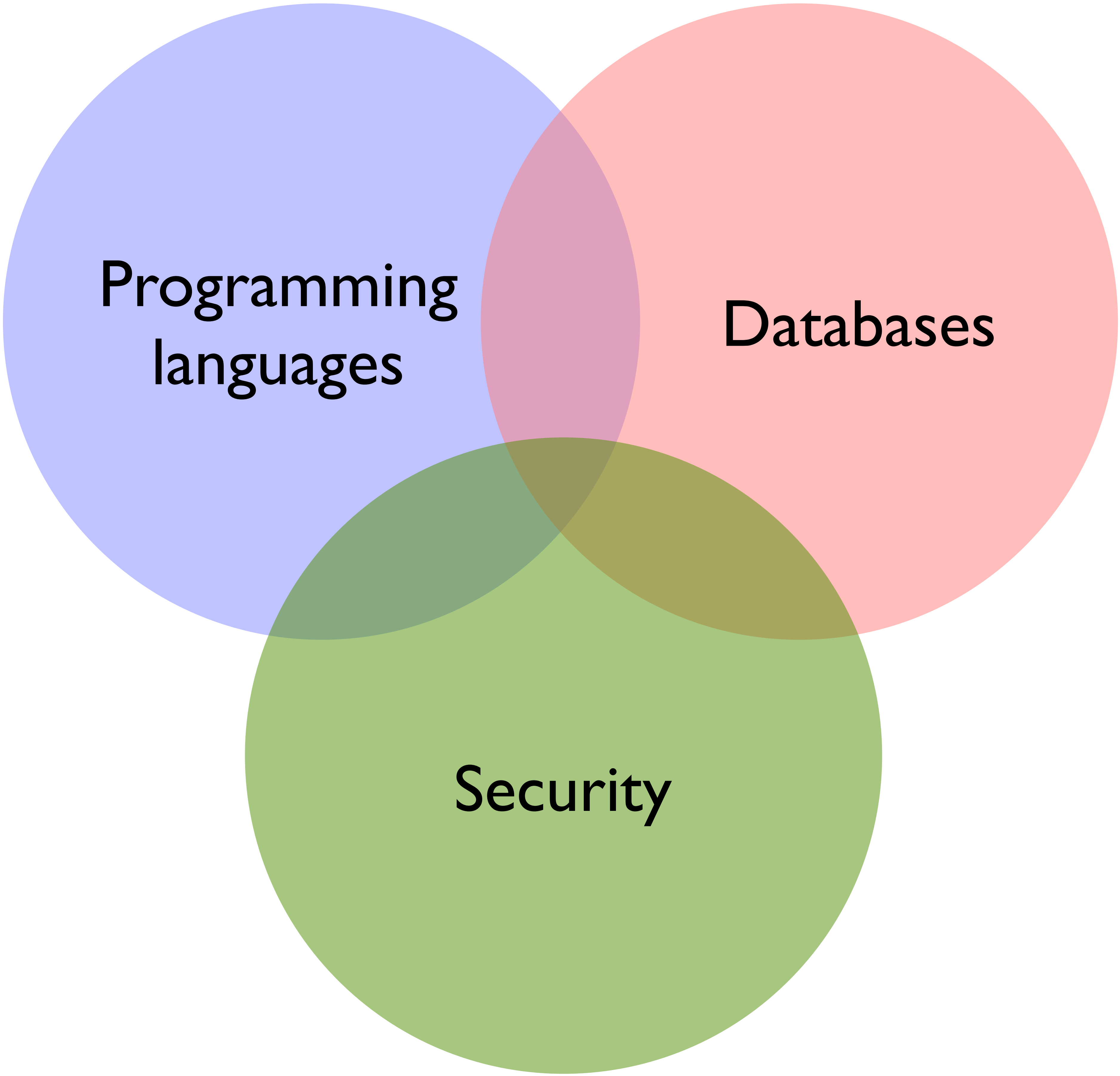
James Cheney
University of Edinburgh
joint work with:
Ghita Berrada, Arthur Chan,
Stefan Fehrenbach, Weili Fu, Rudi Horn,
Roly Perera, Wilmer Ricciotti, Janek Stolarek

Semantics of SQL: quiz

```
SELECT R.A FROM R
EXCEPT
SELECT S.A FROM S
```

$\equiv_?$

```
SELECT R.A FROM R
WHERE R.A NOT IN (
  SELECT S.A FROM S
)
```


Programming languages
Databases
Security

Three themes

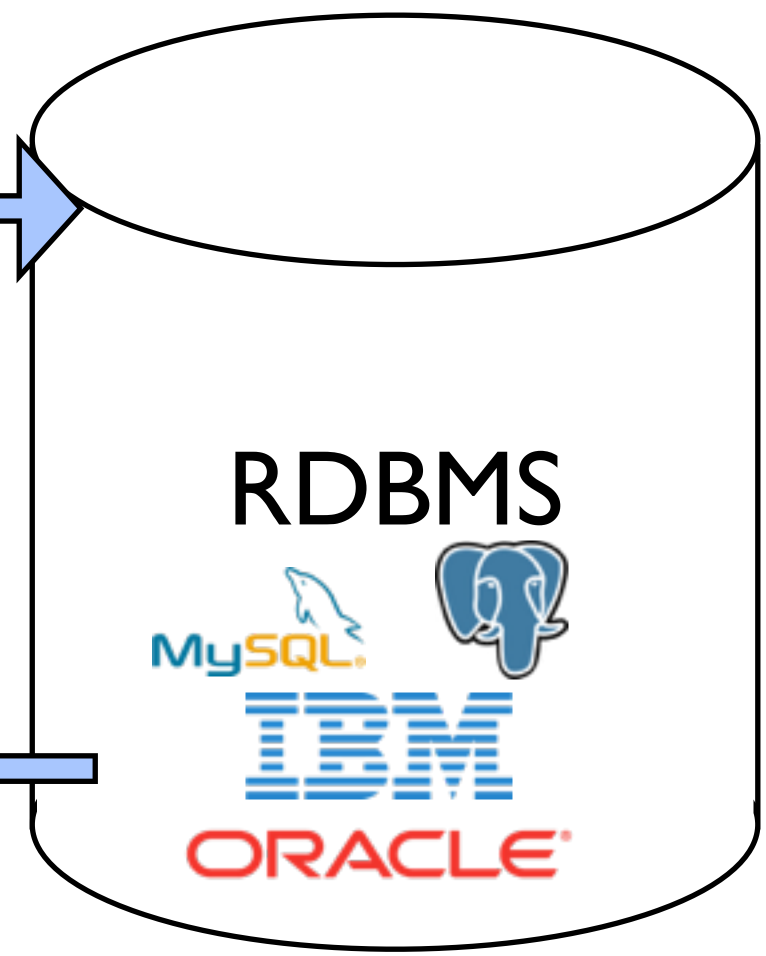
- In roughly chronological order:
  - Language-integrated query
  - Systems provenance and security
  - Towards verified databases
- Common theme: *provenance*
  - that is, metadata about execution / how query results depend on inputs / rich auditable log data
  - (roughly; not really what this talk is about)

Language-integrated query

Databases and Queries

SQL

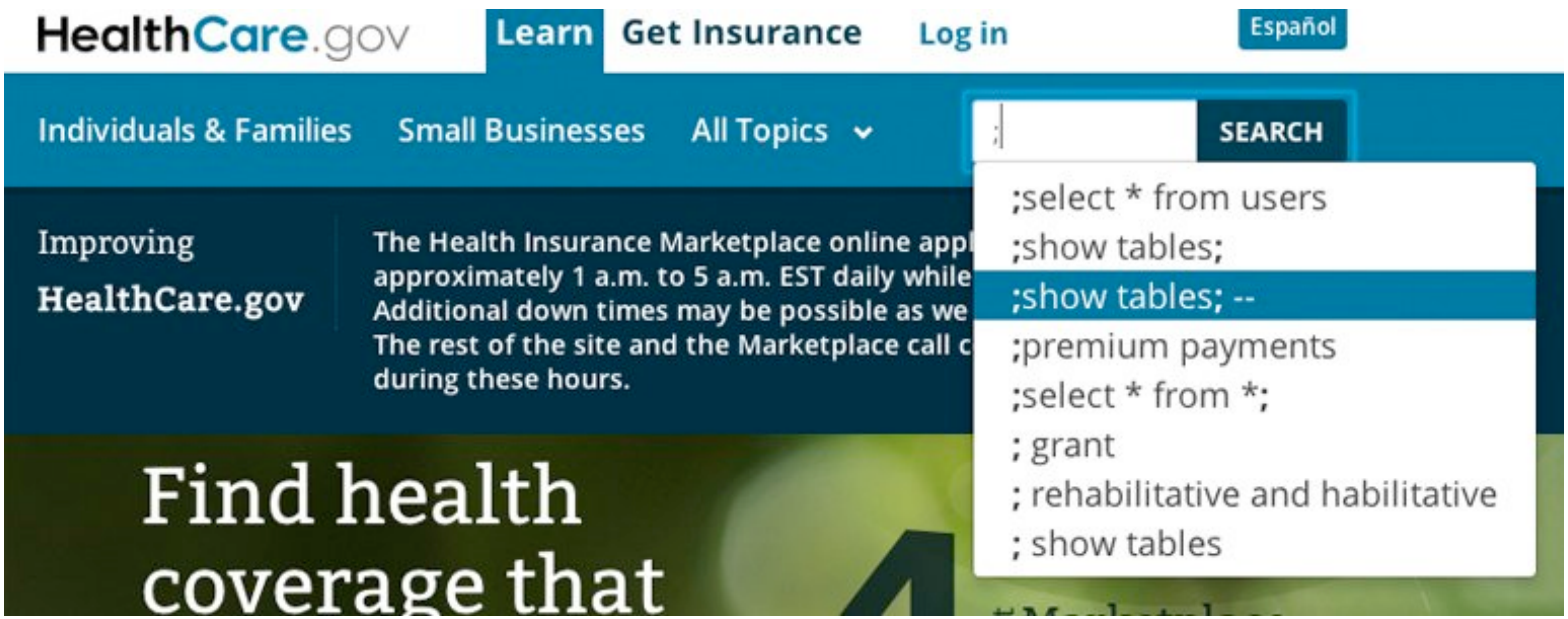
```
SELECT d.dpt, d.id
FROM depts d, emps e
```



Data

dpt	did
Sales	1
Marketing	2
Research	3
...	...

The conventional (JDBC) approach

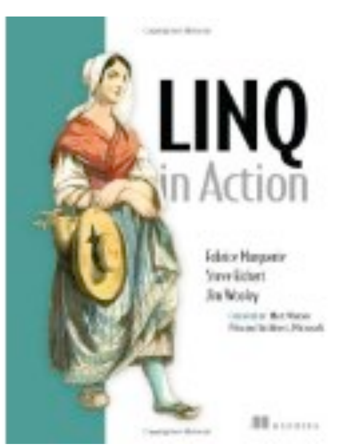


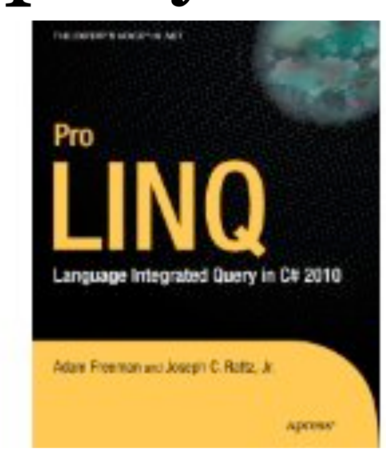
Queries constructed using strings

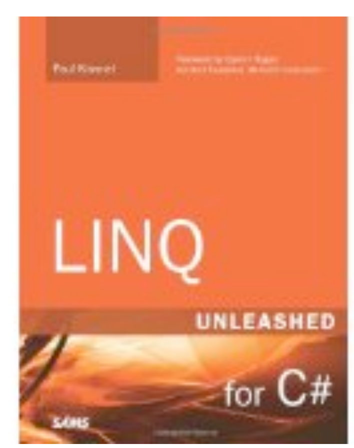
SQL injection attacks can subvert meaning of query

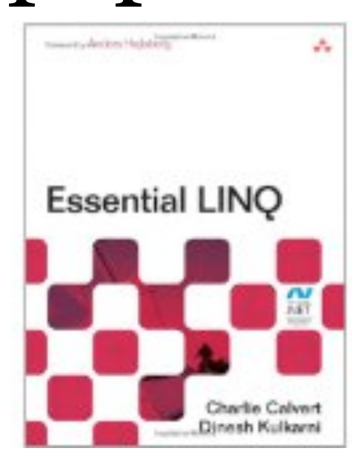
Language-integrated query

- "SQL-like comprehension operations increasingly adopted [in e.g. JavaScript, Python, ...]" - Eric Sedlar, Oracle (SIGMOD 2014 keynote)
  - Microsoft's LINQ and other "language-integrated query" features now popular









Links example

employees

dpt	name	salary
"Product"	"Alex"	40,000
"Product"	"Bert"	60,000
"Research"	"Cora"	50,000
"Research"	"Drew"	70,000
"Sales"	"Erik"	200,000
"Sales"	"Fred"	95,000
"Sales"	"Gina"	155,000

tasks

emp	tsk
"Alex"	"build"
"Bert"	"build"
"Cora"	"abstract"
"Cora"	"build"
"Cora"	"call"
"Cora"	"dissemble"
"Cora"	"enthuse"
"Drew"	"abstract"
"Drew"	"enthuse"
"Erik"	"call"
"Erik"	"enthuse"
"Fred"	"call"
"Gina"	"call"
"Gina"	"dissemble"

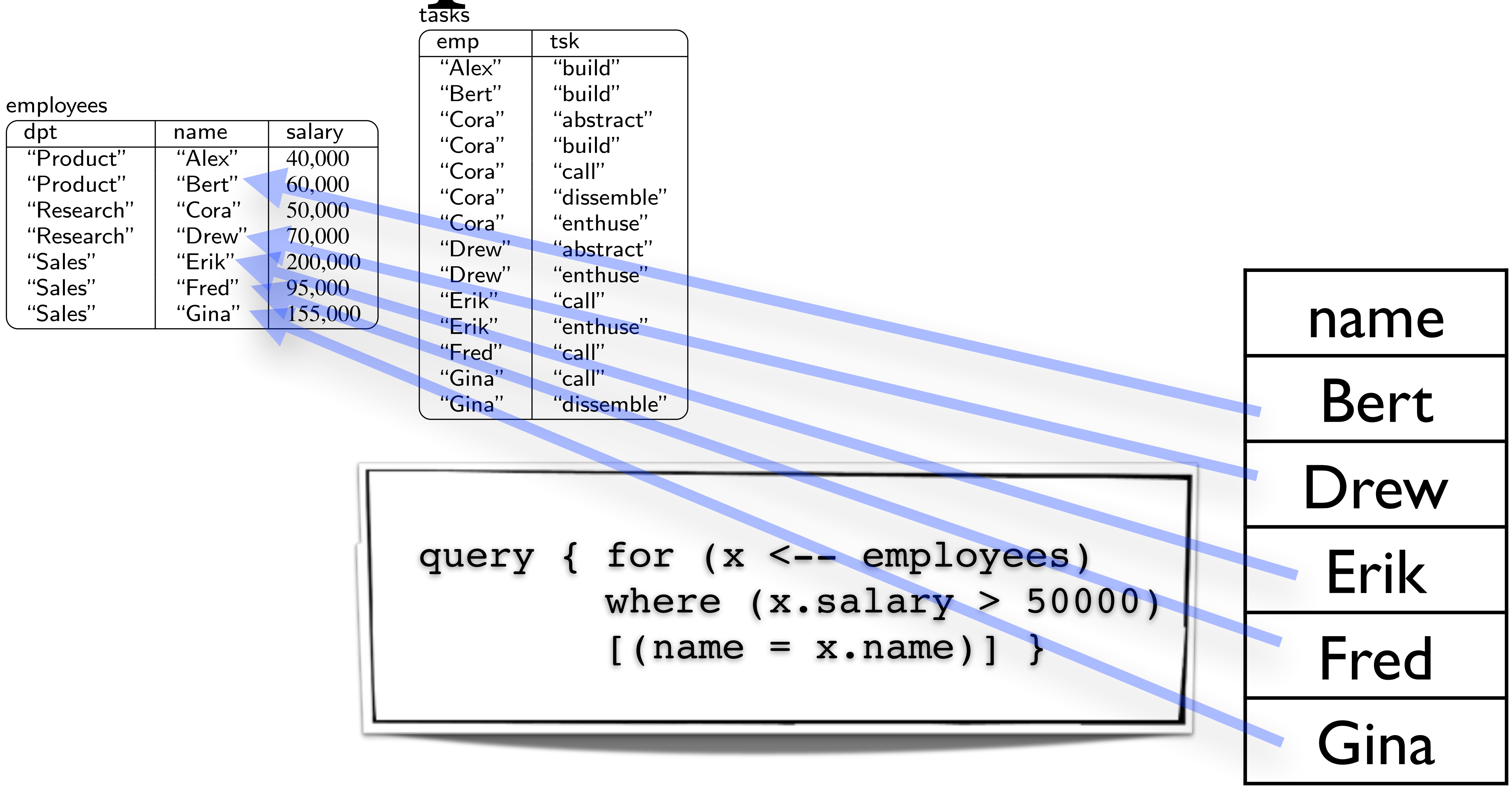
```
query { for (x <-- employees)
        where (x.salary > 50000)
        [(name = x.name)] }
```

name
Bert
Drew
Erik
Fred
Gina

```
select name
from employees e
where e.salary > 50000
```

Wong 2000, Syme 2006, Cooper et al. 2006, 2009

Language-integrated provenance



Fehrenbach & Cheney (PPDP 2016/SCP)

Stolarek et al. (work in progress)

Language-integrated provenance

employees

dpt	name	salary
"Product"	"Alex"	40,000
"Product"	"Bert"	60,000
"Research"	"Cora"	50,000
"Research"	"Drew"	70,000
"Sales"	"Erik"	200,000
"Sales"	"Fred"	95,000
"Sales"	"Gina"	155,000

tasks

emp	tsk
"Alex"	"build"
"Bert"	"build"
"Cora"	"abstract"
"Cora"	"build"
"Cora"	"call"
"Cora"	"dissemble"
"Cora"	"enthuse"
"Drew"	"abstract"
"Drew"	"enthuse"
"Erik"	"call"
"Erik"	"enthuse"
"Fred"	"call"
"Gina"	"call"
"Gina"	"dissemble"

How to generalize?

```
query { for (x <-- employees)
        where (x.salary > 50000)
        [(name = x.name)] }
```

name
Bert
Drew
Erik
Fred
Gina

Fehrenbach & Cheney (PPDP 2016/SCP)

Stolarek et al. (work in progress)

Language-integrated view update

tasks

emp	tsk
“Alex”	“build”
“Bert”	“build”
“Cora”	“abstract”
“Cora”	“build”
“Cora”	“call”
“Cora”	“dissemble”
“Cora”	“enthuse”
“Drew”	“abstract”
“Drew”	“enthuse”
“Erik”	“call”
“Erik”	“enthuse”
“Fred”	“call”
“Gina”	“call”
“Gina”	“dissemble”

employees

dpt	name	salary
“Product”	“Alex”	40,000
“Product”	“Bert”	60,000
“Research”	“Cora”	50,000
“Research”	“Drew”	70,000
“Sales”	“Erik”	200,000
“Sales”	“Fred”	95,000
“Sales”	“Gina”	155,000

```
query { for (x <-- employees)
        where (x.salary > 50000)
        [(name = x.name)] }
```

name
Bert
Drew
Erik
Fred
Gina

Horn (work in progress)

Language-integrated
view update

employees

dpt	name	salary
“Product”	“Alex”	40,000
“Product”	“Bert”	60,000
“Research”	“Cora”	50,000
“Research”	"Vera"	70,000
“Sales”	“Erik”	200,000
“Sales”	“Fred”	95,000
“Sales”	“Gina”	155,000

tasks

emp	tsk
“Alex”	“build”
“Bert”	“build”
“Cora”	“abstract”
“Cora”	“build”
“Cora”	“call”
“Cora”	“dissemble”
“Cora”	“enthuse”
“Drew”	“abstract”
“Drew”	“enthuse”
“Erik”	“call”
“Erik”	“enthuse”
“Fred”	“call”
“Gina”	“call”
“Gina”	“dissemble”

```
query { for (x <-- employees)
        where (x.salary > 50000)
        [(name = x.name)] }
```

name
Bert
Vera
Erik
Fred
Gina

Horn (work in progress)

Language-integrated view update

tasks

emp	tsk
"Alex"	"build"
"Bert"	"build"
"Cora"	"abstract"
"Cora"	"build"
"Cora"	"call"
"Cora"	"dissemble"
"Cora"	"enthuse"
"Drew"	"abstract"
"Drew"	"enthuse"
"Erik"	"call"
"Erik"	"enthuse"
"Fred"	"call"
"Gina"	"call"
"Gina"	"dissemble"

employees

dpt	name	salary
"Product"	"Alex"	40,000
"Product"	"Bert"	60,000
"Research"	"Cora"	50,000
"Research"	"Vera"	70,000
"Sales"	"Erik"	200,000
"Sales"	"Fred"	95,000
"Sales"	"Gina"	155,000

Very subtle in general. Verification?

```
query { for (x <-- employees)
        where (x.salary > 50000)
        [(name = x.name)] }
```

name
Bert
Vera
Erik
Fred
Gina

Horn (work in progress)

Systems provenance & security

npr WAMU 88.5 news arts & life music programs shop

the two-way BREAKING NEWS FROM NPR

AMERICA

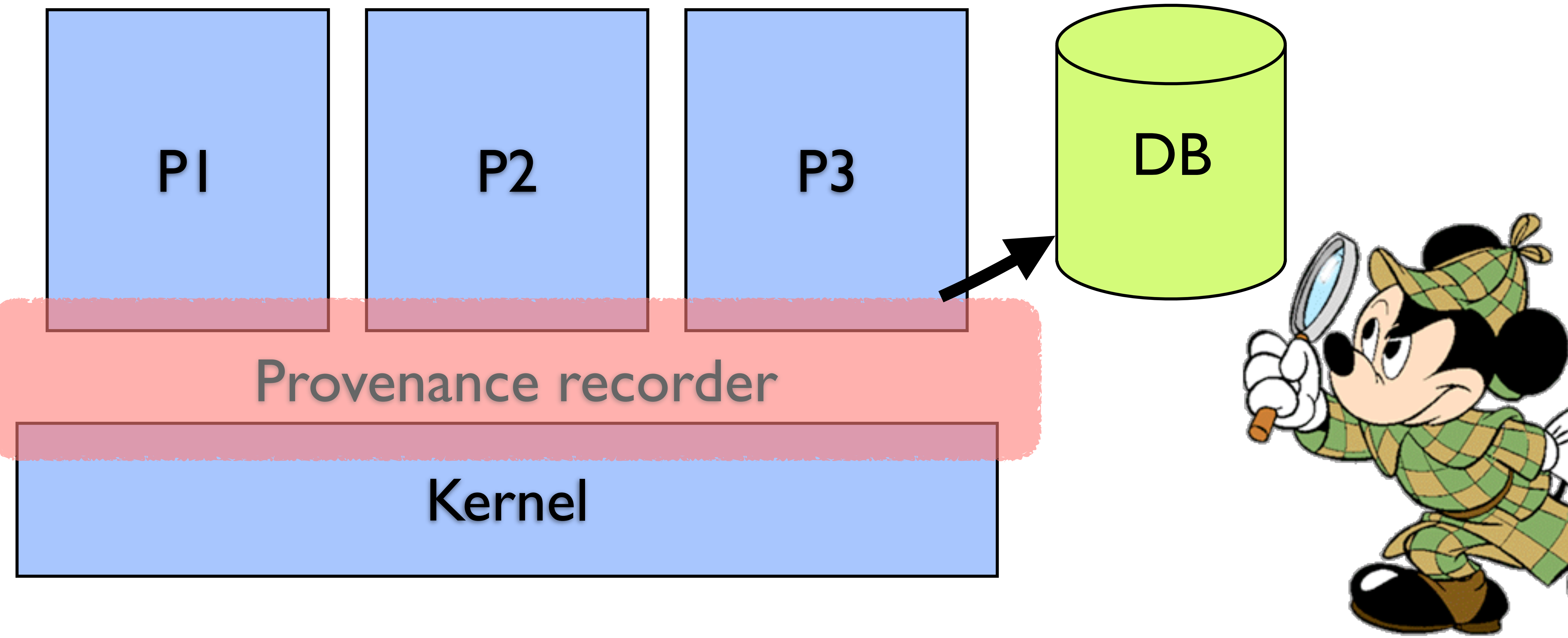
Massive Data Breach Puts 4 Million Federal Employees' Records At Risk

June 4, 2015 · 7:22 PM ET

Office of Personnel Management breach (2015)

DARPA Transparent Computing ($60m,2015-2019)

General idea



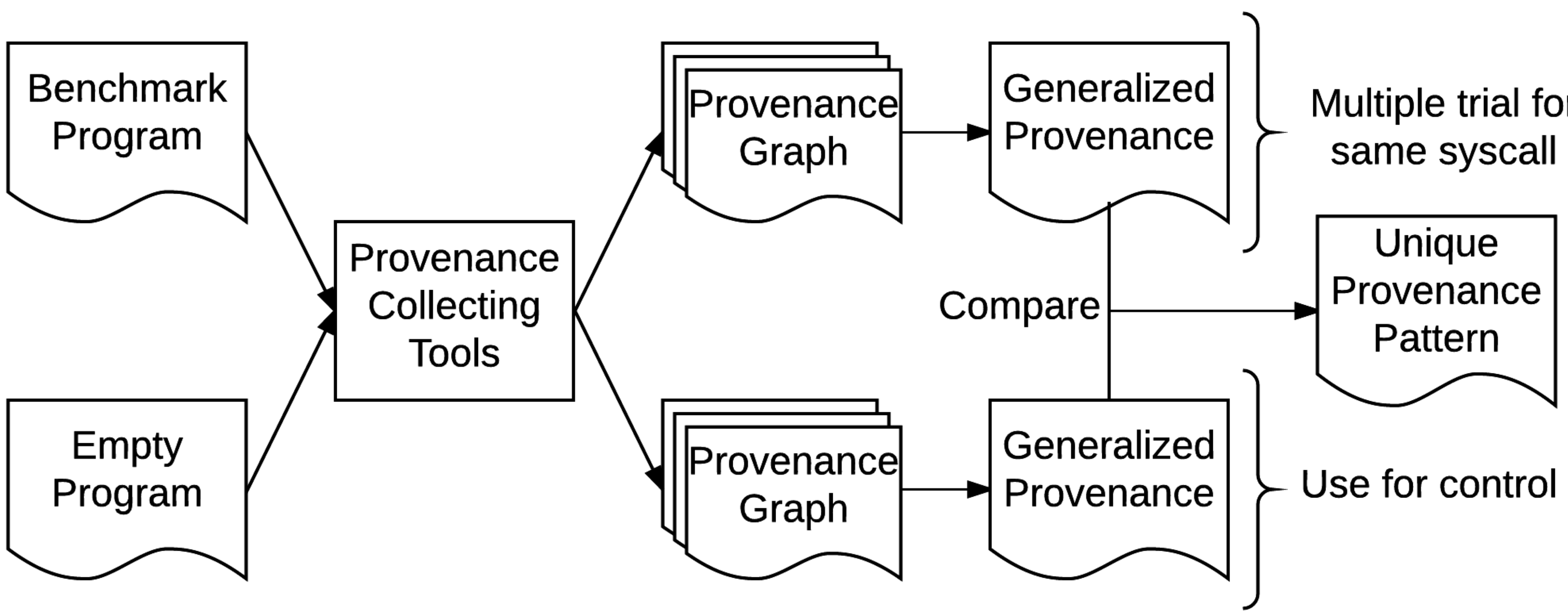
Challenge

- The amount of "normal" system data is massive (up to GBs/day; graphs with millions or billions of nodes/edges)
  - while attacks are ~ 50 nodes/edges.
- We don't know what attacks "look like" in advance
  - We usually don't have annotated data
  - Nor can we expect future attacks to be similar to previously seen ones
- We need **unsupervised** techniques that can find **sparse** anomalies in **large property graphs**
  - this appears to be an open problem in general
- Currently exploring **pattern mining**

Berrada et al., work in progress

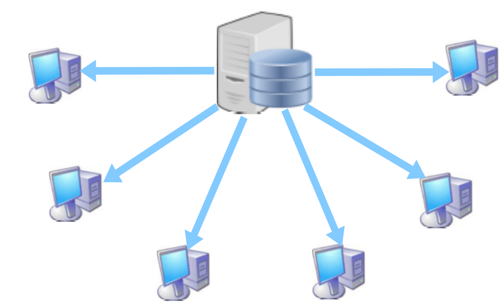
Provenance expressiveness benchmarking

- How do we know correct/sufficient information is recorded?
- How do different recording systems differ?
- Idea: Benchmarking recording systems to observe & classify their behavior *automatically* (at least for small examples)



Chan et al., TaPP 2017

Configuration languages



- High-level *configuration* languages are increasingly popular ("DevOps")
  - Chef, Ansible, **Puppet**
- Configuration errors can have be hard to spot, yet cause massive damage/losses
  - (e.g. $150M cost for recent four hour Amazon outage)
- First step: understanding *semantics* of configuration languages such as Puppet (Fu et al, ECOOP '17)
  - Next: formalizing and implementing provenance tracking for such languages (MSR studentship)

Mechanizing the metatheory of SQL with nulls

Project funded by NCSC/VeTSS
August 2017-March 2018
W. Ricciotti and J. Cheney

Semantics of SQL with nulls: quiz

R

A
1
null

S

A
null

```
SELECT R.A FROM R
EXCEPT
SELECT S.A FROM S
```

≡?

```
SELECT R.A FROM R
WHERE R.A NOT IN (
  SELECT S.A FROM S
)
```

Semantics of SQL with nulls: quiz

R

A
1
null

S

A
null

```
SELECT R.A FROM R
EXCEPT
SELECT S.A FROM S
```

→ {1}

≡?

✗

```
SELECT R.A FROM R
WHERE R.A NOT IN (
  SELECT S.A FROM S
)
```

→ {}

This is because "NOT IN" uses 3-valued semantics...

Semantics of SQL with nulls: quiz

R

A
1
null

S

A
null

```
SELECT R.A FROM
EXCEPT
SELECT S.A FROM S
```

≡?

```
SELECT R.A FROM R
WHERE R.A NOT IN (
  SELECT S.A FROM S
)
```

X

{}

This means DB query optimizers tend to be VERY conservative

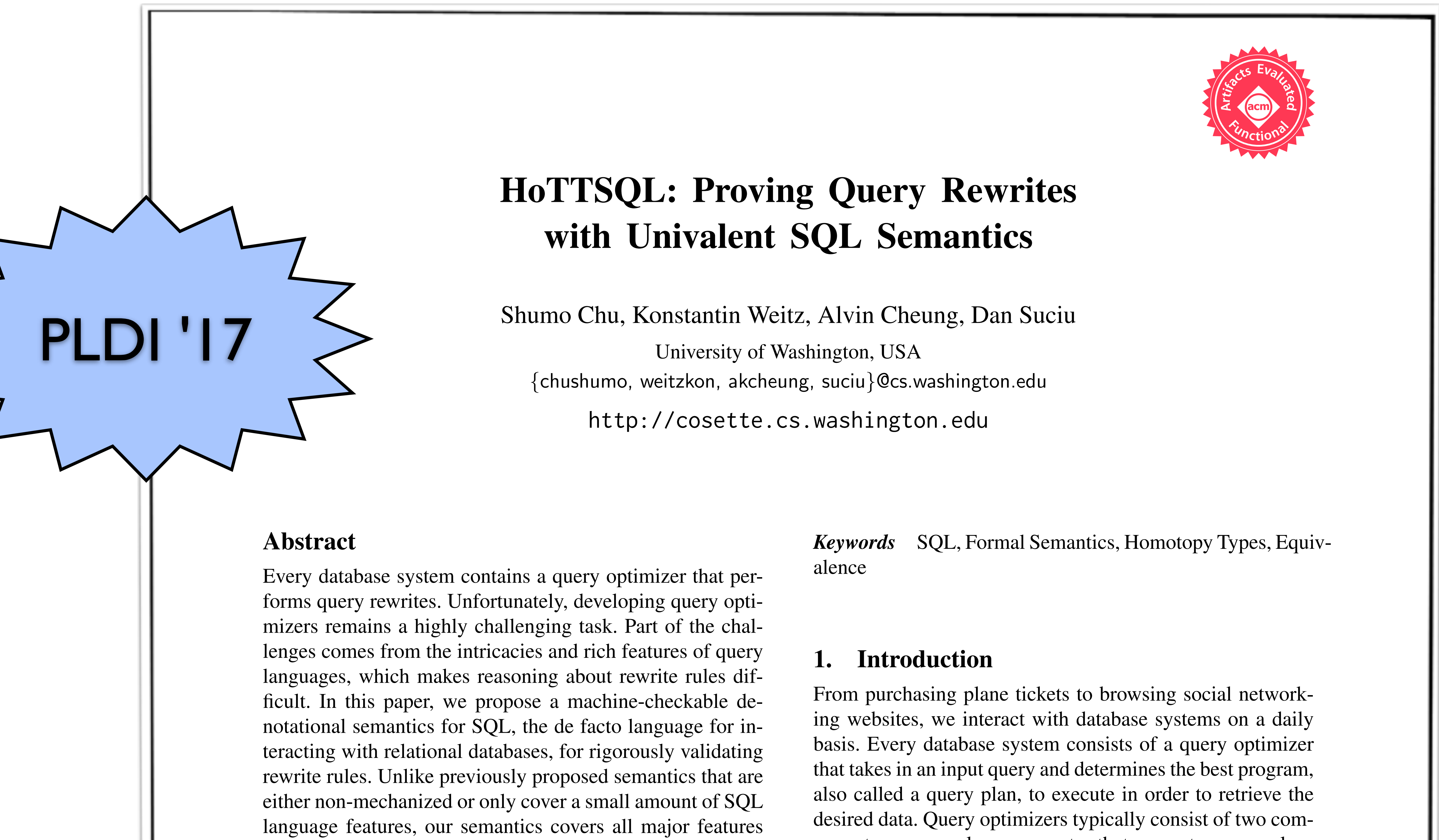
This is because "NOT IN" uses 3-valued semantics...

What is the semantics of SQL?

- It is one of the most widely used and successful "declarative" languages
  - There is even a standard!
- However, its "standard" semantics is (many many pages of) formal-ish English
- To date there is no formal semantics for **all** of SQL
  - Handling complications of "full" SQL such as multiset semantics, grouping, aggregation, **nulls**
  - The "awkward squad" of the database world.

Wouldn't it be nice to formalize that?

(using homotopy type theory, obviously?)



HoTTSQL: Proving Query Rewrites with Univalent SQL Semantics

Shumo Chu, Konstantin Weitz, Alvin Cheung, Dan Suciu
University of Washington, USA
{chushumo, weitzkon, akcheung, suciu}@cs.washington.edu
http://cosette.cs.washington.edu

Abstract

Every database system contains a query optimizer that performs query rewrites. Unfortunately, developing query optimizers remains a highly challenging task. Part of the challenges comes from the intricacies and rich features of query languages, which makes reasoning about rewrite rules difficult. In this paper, we propose a machine-checkable denotational semantics for SQL, the de facto language for interacting with relational databases, for rigorously validating rewrite rules. Unlike previously proposed semantics that are either non-mechanized or only cover a small amount of SQL language features, our semantics covers all major features

Keywords SQL, Formal Semantics, Homotopy Types, Equivalence

1. Introduction

From purchasing plane tickets to browsing social networking websites, we interact with database systems on a daily basis. Every database system consists of a query optimizer that takes in an input query and determines the best program, also called a query plan, to execute in order to retrieve the desired data. Query optimizers typically consist of two com-

Handles most of the "SQL awkward squad"

- "our semantics covers all major features of SQL, including bags, correlated subqueries, aggregation, and indexes"
- combines HoTT with K-relation semantics used for DB provenance, to dramatically simplify query equivalence proofs
- **BUT WAIT....**

3.5 Limitations

*HoTT*SQL does not currently support `ORDER BY`. `ORDER BY` is usually used with `LIMIT` n, e.g., output the first n tuples in a sorted relation. In addition, we currently do not support `NULL`s (i.e., 3-valued logic), and leave them as future work.

Meanwhile, back at the ranch...

with nulls!

VLDB '17

A Formal Semantics of SQL Queries, Its Validation, and Applications

Paolo Guagliardo
School of Informatics
University of Edinburgh
pguaglia@inf.ed.ac.uk

Leonid Libkin
School of Informatics
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

While formal semantics of theoretical languages underlying SQL have been provided in the past, they all made simplifying assumptions ranging from changes in the syntax to omitting bag semantics and nulls. This situation is reminiscent of what happens in the field of programming languages, where semantics of formal calculi underlying main features of languages are abundant, but formal semantics of real languages that people use are few and far between.

We take the basic class of SQL queries – essentially SELECT-FROM-WHERE queries with subqueries, set/bag operations, and nulls – and define a formal semantics for it, without any departures from the real language. Already this fragment requires decisions related to the data model and

as it needs to account for all its idiosyncrasies. This has been done for several languages [1, 13, 18, 25, 28, 29]; the difference is that to describe such a formal semantics one needs a book, rather than a paper (or sometimes even a book to explain what the first book said [24]).

When it comes to the main query language used by relational DBMSs – SQL – we have the Standard [20], but it cannot serve as a formal semantics, as it is written in natural language. In fact, it is well known that different vendors of RDBMSs interpret various points of the Standard differently (see, e.g., [4, 21]). A natural language description does not lend itself to proper formal reasoning that is necessary to derive language equivalences and optimization rules.

Given the problems of using the Standard as the definition

Our project

- Formalize Guagliardo & Libkin semantics of (subset of) SQL with nulls...
  - using "conventional" Coq formalization approach, at least initially
- Try to reconcile with Chu et al.'s HoTTSQL approach
  - also: consider the "adequacy" of HoTT / K-relation interpretation of SQL
- Goal: first full formalization of "real" SQL with nulls
  - + verification or counterexamples to equivalences

Conclusion

- My (group's) research covers a range of topics
  - Programming languages + DB = language integrated query
  - Security + DB = provenance mining
  - Verification + DB = mechanizing metatheory of SQL
- Long-term vision: **verified trustworthy database systems**
  - that provide answers that are ***correct*** (queries executed correctly)
  - and ***trustworthy*** (provenance/explanation of how results were derived)