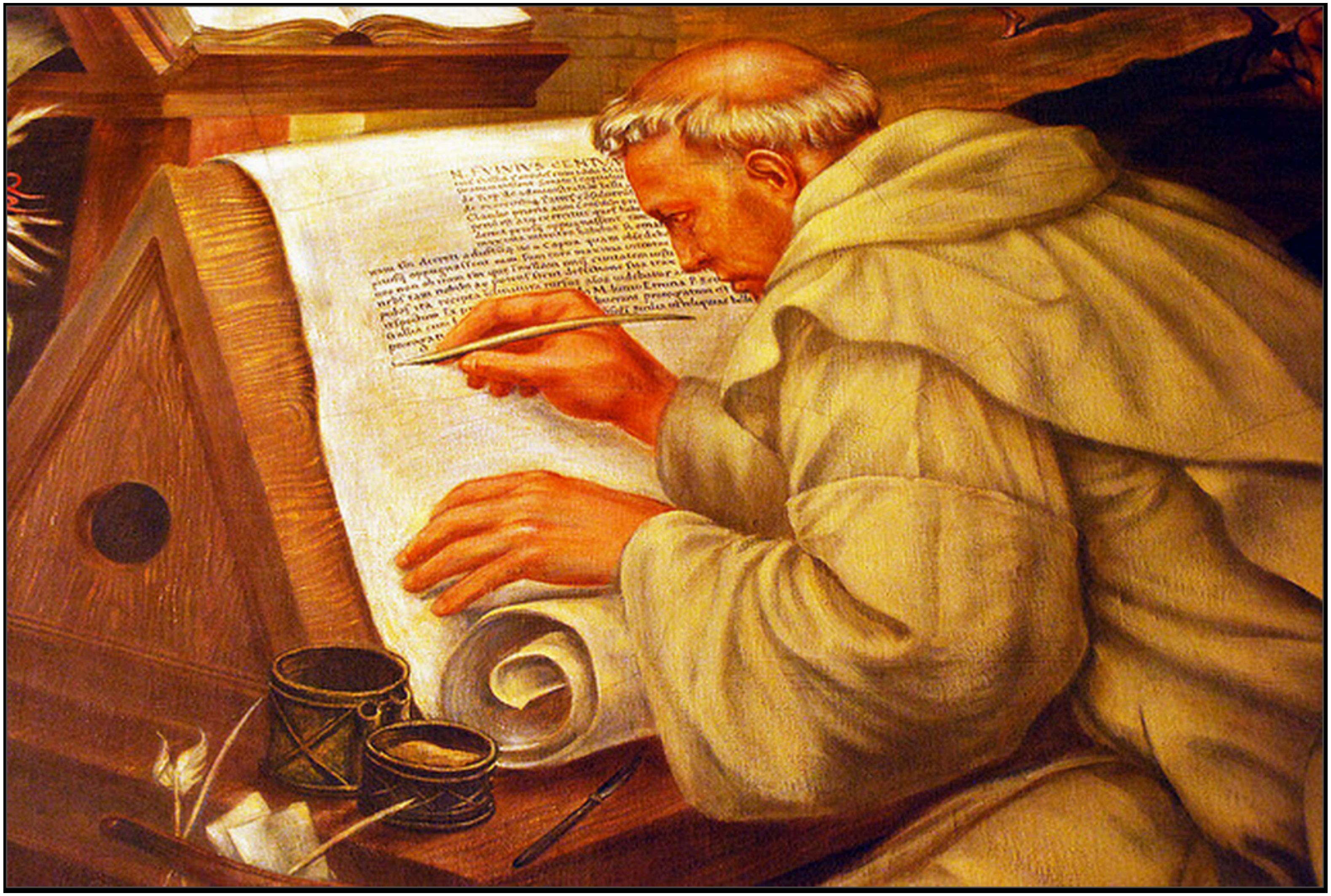
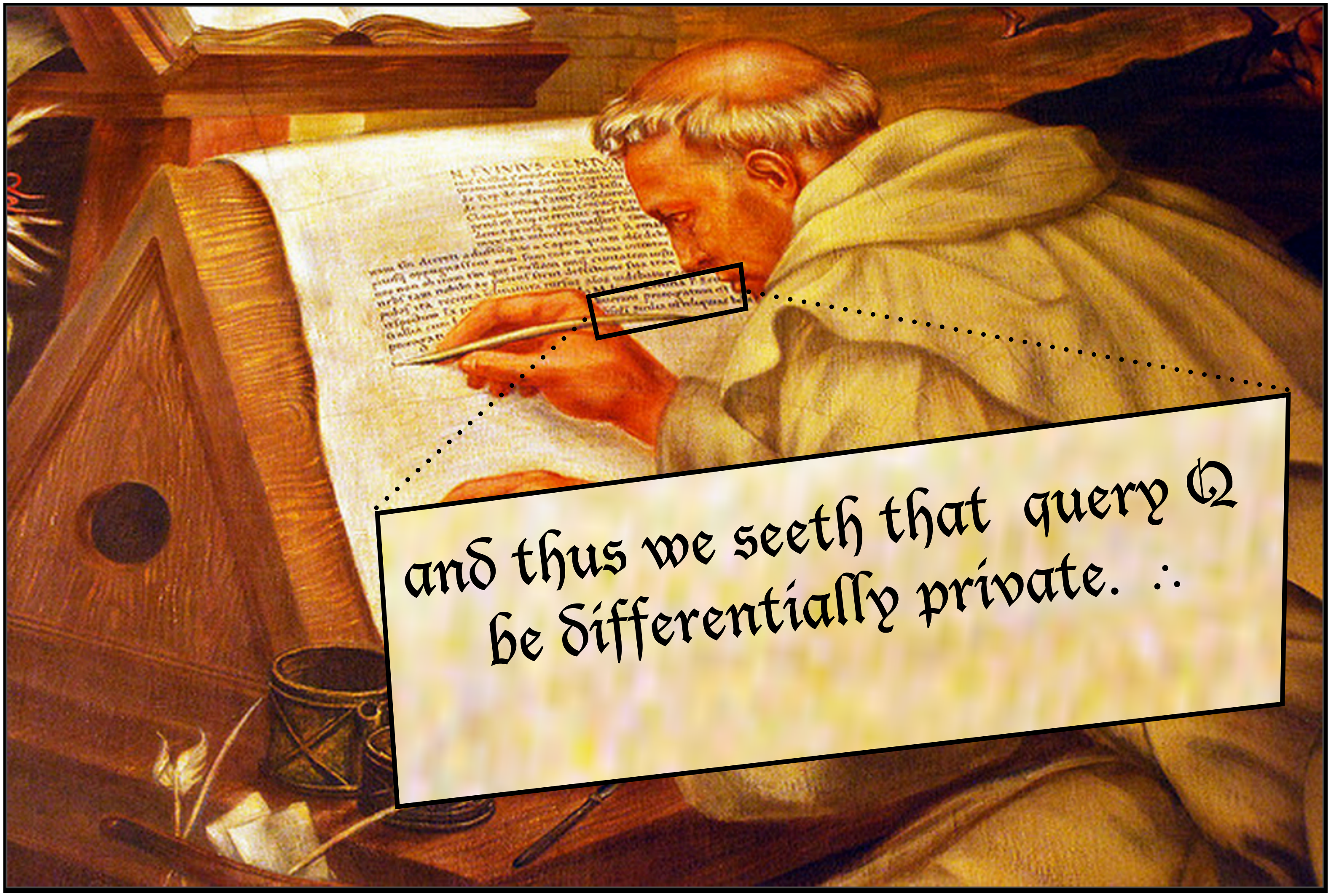
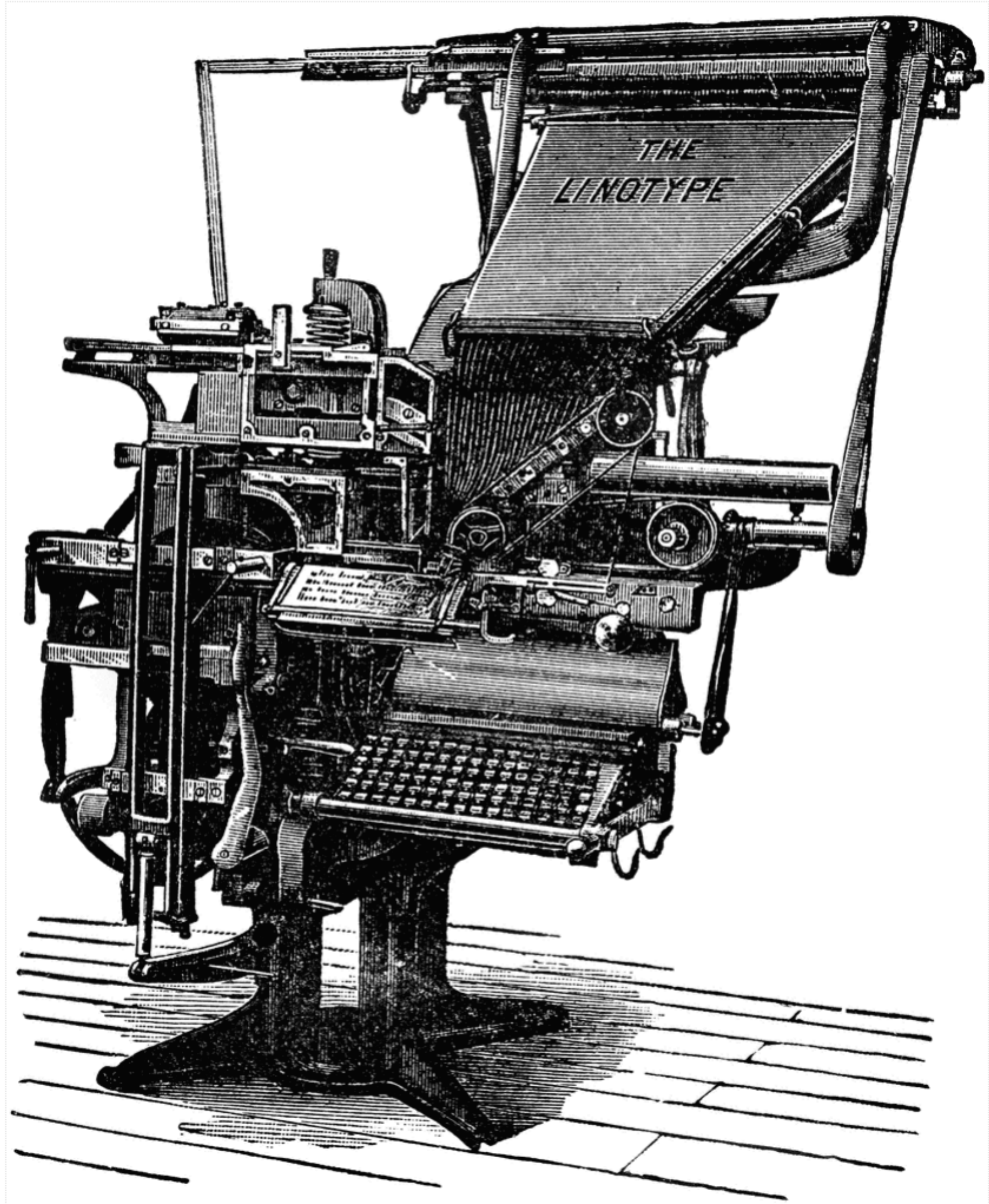
Differential Privacy in the Programming Languages Community

Benjamin C. Pierce
University of Pennsylvania

DIMACS
October 2012




and thus we seeth that query Q
be differentially private. ∴


THE
LINOTYPE

PL for DP

- Goal:
    - Provide tools for expressing queries so that their privacy can be *mechanically verified*
- Tools:
    - compositionality
    - little languages
    - type systems
    - proof checkers

Dynamic tracking

- least work
- medium benefit
- very flexible

Static analysis

- little work
- high benefit
- less flexible

Machine-checked proof

- *lots* of work
- high benefit
- very flexible

Outline

- Dynamic approaches (PINQ, Airavat)
- Static checking (Fuzz, DFuzz)
- Machine-checked proof (CertiPriv)

- Other PL work
- Other systems work

briefly...

Any questions?

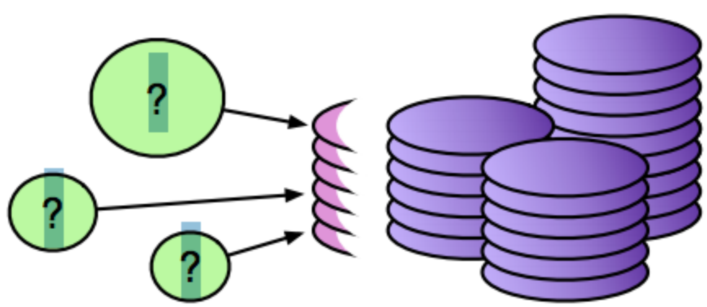
Dynamic tracking

PINQ

Frank McSherry, *Privacy Integrated Queries*, SIGMOD, 2009
Frank McSherry, *Privacy Integrated Queries*, Communications of the ACM, 2010
Frank McSherry and Ratul Mahajan, *Differentially-Private Network Trace Analysis*, SIGCOMM 2010,

Privacy INtegrated Queries

- Common platform for differentially private data analyses.
  - Provides interface to data that looks very much like LINQ (C#'s "language-integrated queries")
  - All access through the interface is guaranteed to be differentially private
- (Non-privacy-expert) data analysts write arbitrary LINQ code against data sets in C#.



```
var data = new PINQueryable<SearchRecord>(... ...);

var users = from record in data
            where record.Query == argv[0]
            groupby record.IPAddress;

Console.WriteLine(argv[0] + “:” + users.Count(0.1));
```

How it works...

- Each private data source is wrapped in a `PINQueriable` object, which is responsible for...
  - mediating accesses to the underlying data
  - remembering how much privacy budget is left
  - deducting from the budget whenever an aggregation operator is applied to this PINQueriable object (or any other one derived from it)
  - denying access once the budget is exhausted

Aggregation operations

- **NoisyCount**
  - arguments: a PINQueriable and a desired accuracy for the count
  - calculates how much privacy budget must be expended to produce an answer with this accuracy
    - asks PINQueriable to deduct this much
  - returns count plus appropriate Laplace noise
- Similarly: NoisySum, NoisyAverage, etc.

Transformations

- Each transformation method...
  - maps a PINQueriable to one or more new PINQueriables...
  - that, when later aggregated, will *forward* the privacy cost to the original object...
  - after applying an appropriate scale factor (i.e., after taking account of the sensitivity of the transformation).

Transformations

- **Where**: takes a predicate and returns a new PINQueriable wrapping the subset of the data satisfying the predicate
- **GroupBy**: takes a function mapping records to key values, and results in a list of groups: for each observed key, the group of records that map to that key
- **Join**: takes two data sets, key selection functions for each, and returns the list of all pairs of elements whose keys match (to prevent blowup in the size of the output, each input data set is first grouped by its join keys, so that each join key becomes a primary key)
- **Partition**: like GroupBy, but must be explicitly provided with a list of candidate keys; its result is a list of PINQueryable objects, one for each candidate key, containing the (possibly empty) subset of records with this key.

Evaluation

- Advantages of the PINQ approach:
  - Simple to implement and explain
  - Flexible: wide range of DP queries can be expressed
- Limitation: No static checking → privacy budget violations only detected at the end
  - may waste time or privacy if a long-running privacy-demanding computation needs more budget than is available

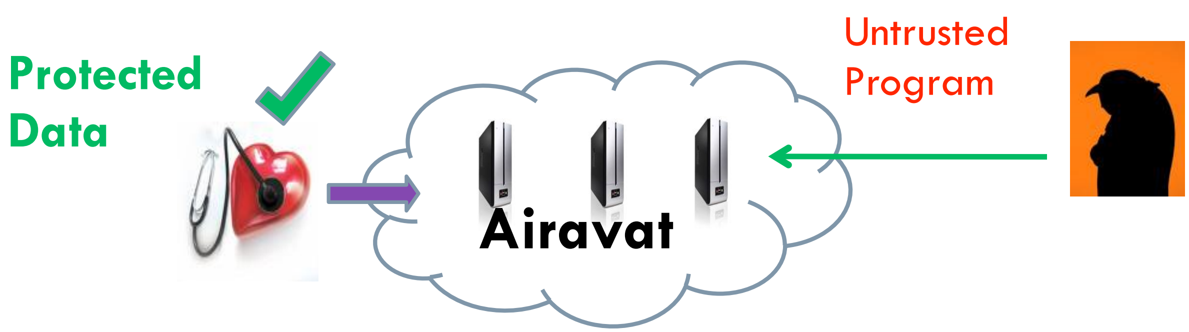
Airavat

Roy, Setty, Kilzer, Shmatikov, & Witchel. *Airavat: Security and privacy for MapReduce*. NSDI, 2010

(Thanks to Roy for slides!)

Airavat

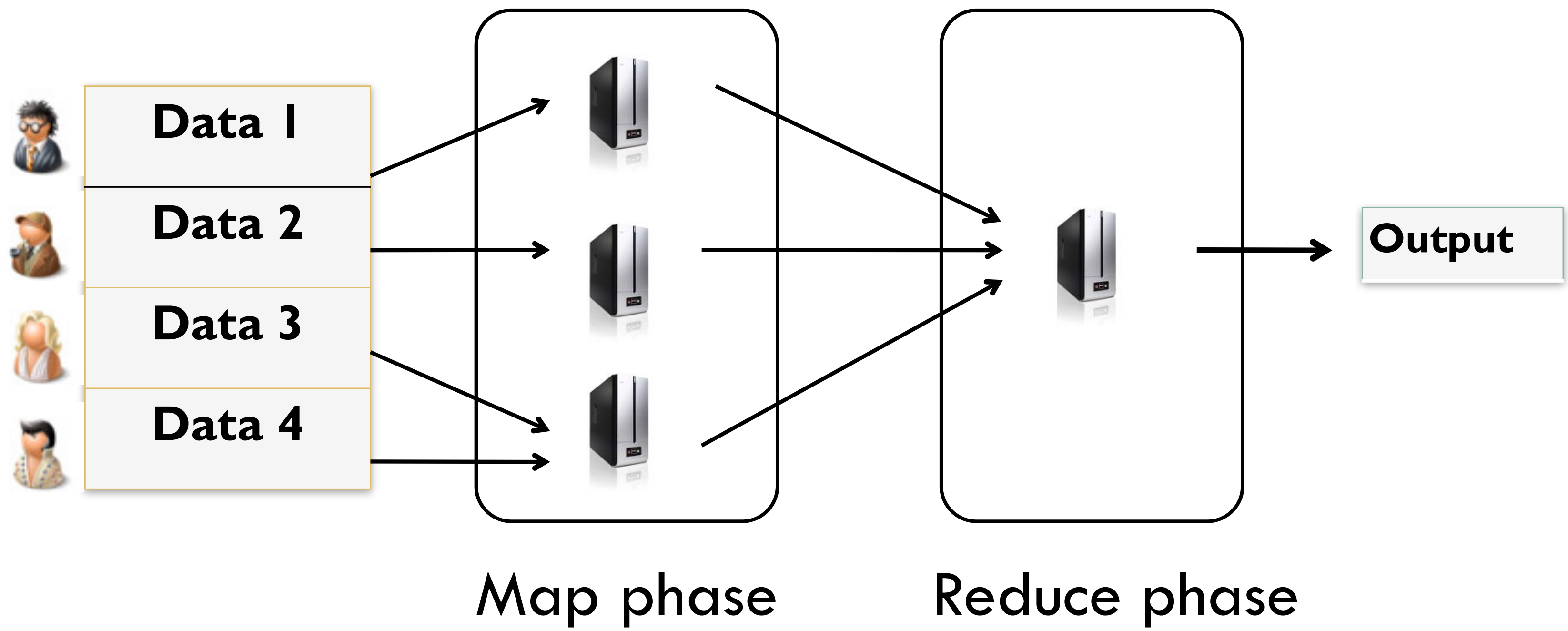
Framework for privacy-preserving MapReduce computations with untrusted code.



Airavat is the elephant of the clouds (Indian mythology).

Background: MapReduce

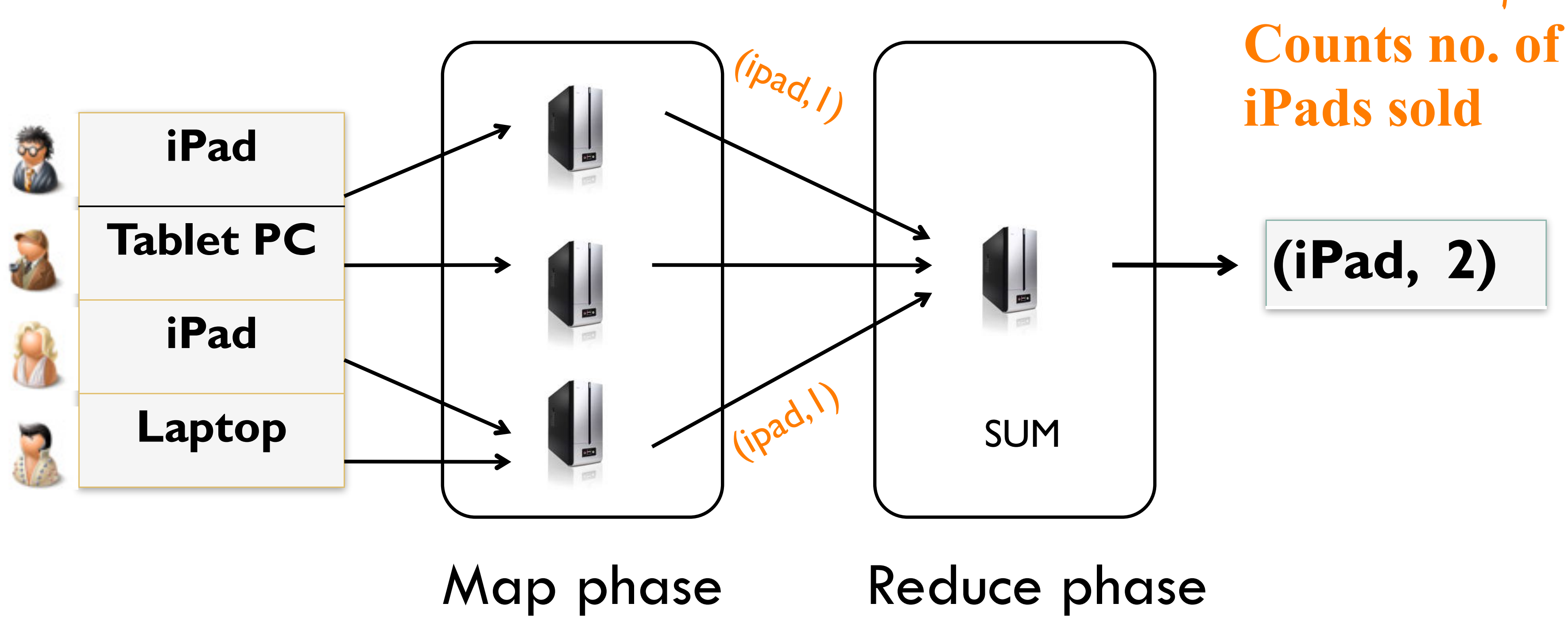
$$map(k_1,v_1) \rightarrow \quad list(k_2,v_2)$$
$$reduce(k_2, list(v_2)) \rightarrow \quad list(v_2)$$



MapReduce example

Map(input)→{ if (input has iPad) print (iPad, 1) }

Reduce(key, list(v))→{ print (key + ","+ SUM(v)) }

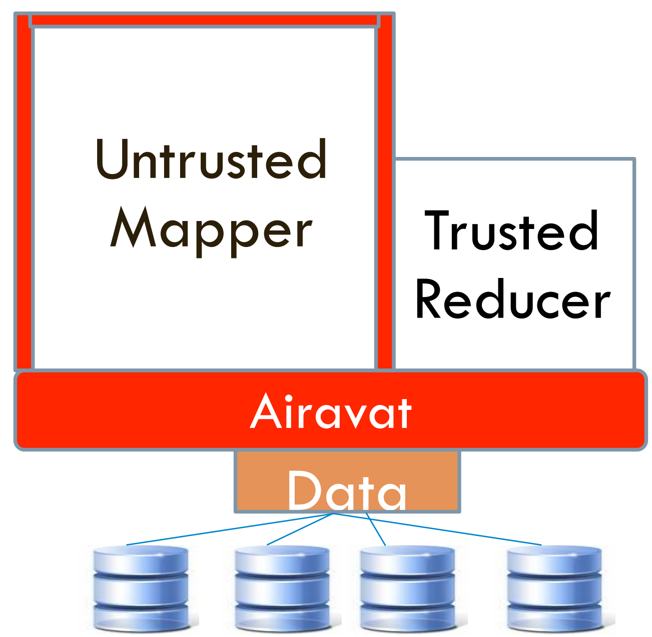


Programming model

Split MapReduce into untrusted mapper + trusted reducer

Limited set of stock reducers

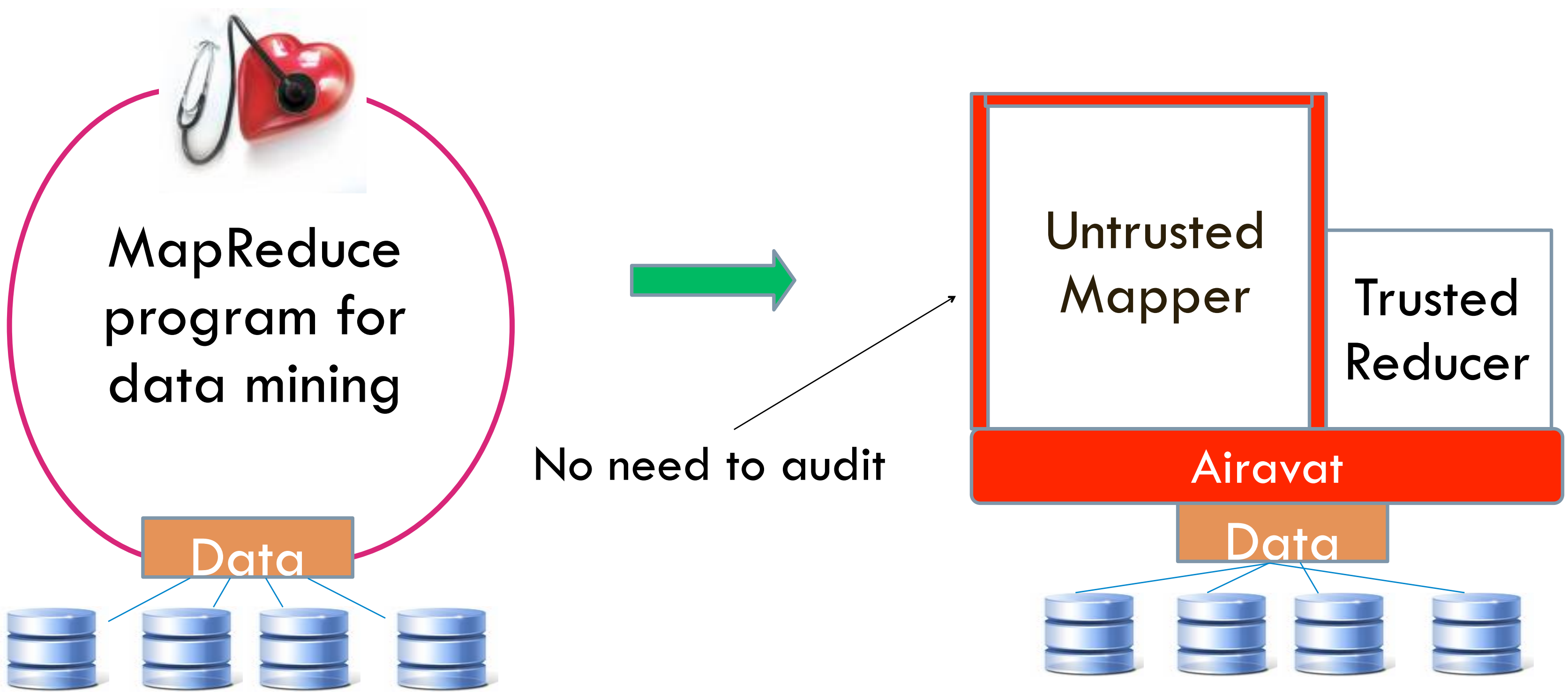
No need to audit



Programming model

Need to confine the mappers !

Guarantee: Protect the privacy of data providers

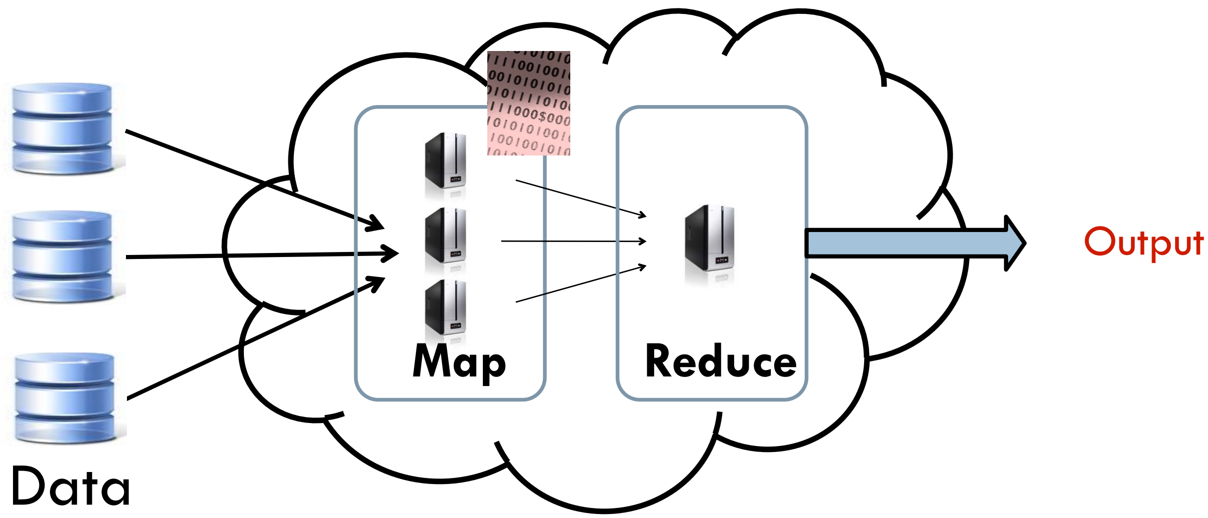


Airavat mechanisms

Mandatory access control + Differential privacy

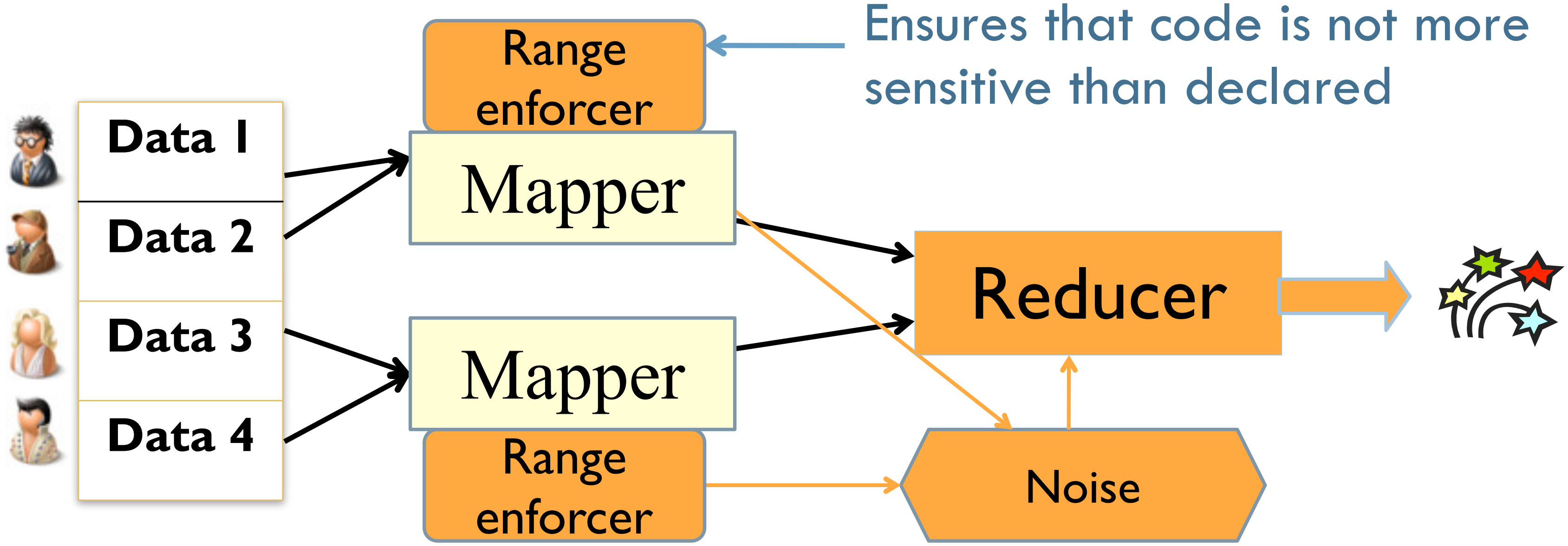
Prevent leaks through storage channels like network connections, files…

Prevent leaks through the output of the computation



Enforcing differential privacy

- Malicious mappers may output values outside the range
- If a mapper produces a value outside the range, it is replaced by a value inside the range
  - User not notified… otherwise possible information leak



Static Analysis

Motivation

- Want to know *in advance* how much privacy a query will use
  - Dynamic tracking of privacy depletion gives us no well-grounded way of looking at a program and *predicting* its privacy cost

Fuzz

Reed and Pierce, *Distance makes the types grow stronger: A calculus for differential privacy*. ICFP 2010.

See also: Palamidessi and Stronati: *Differential Privacy for Relational Algebra: Improving the Sensitivity Bounds via Constraint Systems,* QAPL 2012

Fuzz

- Higher-order functional language (ML-like)
- Static type system features
  - sensitivity tracking based on *linear logic*
  - internalized type of probability distributions
- Differential privacy guaranteed by typechecking

Sensitivity -> Privacy

+

PL for Sensitivity

=

PL for Privacy

Quick example

Suppose we have the following functions:

$$\textit{over\_40} : \text{row} \rightarrow \text{bool}$$
$$\textit{size} : \text{db} \multimap \mathbb{R}$$
$$\textit{filter} : (\text{row} \rightarrow \text{bool}) \rightarrow \text{db} \multimap \text{db}$$
$$\textit{add\_noise} : \mathbb{R} \multimap \bigcirc\mathbb{R}$$

This expression computes a differentially private count of database rows satisfying *over_40*:

$$\lambda d : \text{db}.\ \textit{add\_noise}\,(\textit{size}\,(\textit{filter}\ \textit{over\_40}\ d)) \ \ : \text{db} \multimap \bigcirc\mathbb{R}$$

Punchline

Typing guarantees differential privacy

Theorem If e is a closed program with $\vdash e : \tau \multimap \bigcirc\sigma$ is an ϵ-differentially private function from τ to σ.

Metrics

Main idea: For every type τ, define a metric d_τ

Definition A function $f : \tau_1 \to \tau_2$ is c-*sensitive* iff

$$d_{\tau_2}(f(x), f(y)) \leq c \cdot d_{\tau_1}(x, y)$$

for all $x, y \in \tau_1$

Base type

The primitive type $\mathbb{R}$ has the usual metric:

$$d_{\mathbb{R}}(x, y) = |x - y|$$

Sets

τ set is a type for each type τ, with Hamming metric

$$d_{\tau\,\mathsf{set}}(S_1, S_2) = ||S_1 \triangle S_2||$$

Primitives:

$\textit{size} : \tau\ \mathsf{set} \multimap \mathbb{R}$

$\textit{setfilter} : (\tau \to \mathsf{bool}) \to \tau\ \mathsf{set} \multimap \tau\ \mathsf{set}$

$\textit{setmap} : (\sigma \to \tau) \to \tau \to \sigma\ \mathsf{set} \multimap \tau\ \mathsf{set}$

$\cap, \cup, \setminus : \tau\ \mathsf{set} \otimes \tau\ \mathsf{set} \multimap \tau\ \mathsf{set}$

$\textit{split} : (\tau \to \mathsf{bool}) \to \tau\ \mathsf{set} \multimap \tau\ \mathsf{set} \otimes \tau\ \mathsf{set}$

Scaling

For each type τ, let $!_r\tau$ be the type with the same values as τ, but with the metric 'scaled up' by r:

$$d_{!_r\tau}(x, y) = r \cdot d_\tau(x, y)$$

Proposition A function f is a c-sensitive function in $\tau_1 \to \tau_2$ iff it is a 1-sensitive function in $!_c\tau_1 \to \tau_2$.

Pairs

$\tau_1 \otimes \tau_2$ is the type of pairs (v_1, v_2) with $v_1 \in \tau_1$ and $v_2 \in \tau_2$.

Distance between pairs is the sum of the distances between components:

$$d_{\tau_1 \otimes \tau_2}((v_1, v_2), (v_1', v_2)) = d_{\tau_1}(v_1, v_1') + d_{\tau_2}(v_2, v_2')$$

Examples

1-sensitive functions in $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R}$:

$$f_1(x, y) = x + y \qquad f_2(x, y) = x - y$$

1-sensitive functions in $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R} \otimes \mathbb{R}$:

$$f_3(x, y) = (x, y) \qquad f_4(x, y) = (y, x)$$

$$f_5(x, y) = (x + y, 0) \qquad \textit{cswp}(x, y) = \begin{cases} (x, y) & \text{if } x < y \\ (y, x) & \text{otherwise} \end{cases}$$

Non-example:

$$f_6(x, y) = (x, x)$$

is *not* a 1-sensitive function in $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R} \otimes \mathbb{R}$.

Another metric for pairs

$\tau_1 \,\&\, \tau_2$ consists of pairs $\langle v_1, v_2 \rangle$ with the metric

$$d_{\tau_1 \& \tau_2}(\langle v_1, v_2 \rangle, \langle v_1', v_2 \rangle) = \max(d_{\tau_1}(v_1, v_1'), d_{\tau_2}(v_2, v_2'))$$

E.g.

$$f_7(x, y) = \langle x, x \rangle$$

is a 1-sensitive function $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R} \,\&\, \mathbb{R}$

Proposition If $f : \tau \to \tau_1$ and $g : \tau \to \tau_2$ are c-sensitive, then $\lambda x. \langle f\ x, g\ x \rangle$ is a c-sensitive function in $\tau \to \tau_1 \,\&\, \tau_2$.

Functions

$\tau_1 \multimap \tau_2$ is the type of 1-sensitive functions $f : \tau_1 \to \tau_2$

$$d_{\tau_1 \multimap \tau_2}(f, f') = \max_{x \in \tau_1} d_{\tau_2}(f(x), f'(x))$$

$!_c \tau_1 \multimap \tau_2$ is c-sensitive functions from τ_1 to τ_2

Disjoint unions

$\tau_1 + \tau_2$ is the disjoint union of τ_1 and τ_2

Values: $\{\mathbf{inj}_1\, v \mid v \in \tau_1\} \cup \{\mathbf{inj}_2\, v \mid v \in \tau_2\}$

Metric:

$$d_{\tau_1+\tau_2}(v, v') = \begin{cases} d_{\tau_1}(v_0, v_0') & \text{if } v = \mathbf{inj}_1\, v_0 \text{ and } v' = \mathbf{inj}_1\, v_0'; \\ d_{\tau_2}(v_0, v_0') & \text{if } v = \mathbf{inj}_2\, v_0 \text{ and } v' = \mathbf{inj}_2\, v_0'; \\ \infty & \text{otherwise.} \end{cases}$$

Booleans

This metric defines an *extremely* disjoint union of two components. E.g., for the type

$$\text{bool} = 1 + 1$$

we have

$$d(\textit{true}, \textit{true}) = d(\textit{false}, \textit{false}) = 0$$

$$d(\textit{true}, \textit{false}) = d(\textit{false}, \textit{true}) = \infty$$

Upshot: Easy to write c-sensitive functions *from* bool to other types, but hard for a nontrivial function *to* bool to be c-sensitive. E.g.,

$$\textit{gtzero} : \mathbb{R} \to \text{bool}$$

is not c-sensitive for any finite c.

Lists

Two lists of different lengths are at distance ∞ from each other (this corresponds to the definition of the metric on disjoint union types).

For two lists $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$ of the same length,

$$d_{\tau\,\mathsf{list}}([x_1, \ldots, x_n], [y_1, \ldots, y_n]) = \sum_{i=1}^{n} |x_i - y_i|.$$

Sorting

Can't have this:

$$\geq_{\mathbb{R}} : \mathbb{R} \otimes \mathbb{R} \multimap \text{bool}$$

So use this:

$$\textit{cswp} : \mathbb{R} \otimes \mathbb{R} \multimap \mathbb{R} \otimes \mathbb{R}$$

Now:

$$\begin{aligned}
&\textit{insert} : \mathbb{R} \multimap \mathbb{R}\,\text{list} \multimap \mathbb{R}\,\text{list} \\
&\textit{insert}\; x\; [\,] = [x] \\
&\textit{insert}\; x\; (h :: tl) = \mathbf{let}(a, b) = \textit{cswp}\; (x, h)\; \mathbf{in} \\
&\qquad\qquad\qquad\qquad\qquad a :: (\textit{insert}\; b\; tl)
\end{aligned}$$

$$\begin{aligned}
&\textit{sort} : \mathbb{R}\,\text{list} \multimap \mathbb{R}\,\text{list} \\
&\textit{sort}\; [\,] = [\,] \\
&\textit{sort}\; (h :: tl) = \textit{insert}\; h\; (\textit{sort}\; tl)
\end{aligned}$$

Sensitivity -> Privacy

+

PL for Sensitivity

=

PL for Privacy

But wait, there's more...

Probability distributions

For each type τ, let $\bigcirc\tau$ be the type of *probability distributions over* τ, with

$$d_{\bigcirc\tau}(\delta_1, \delta_2) = \frac{1}{\epsilon}\left(\max_{x \in \tau}\left|\ln\left(\frac{\delta_1(x)}{\delta_2(x)}\right)\right|\right)$$

add_noise : $\mathbb{R} \multimap \bigcirc\mathbb{R}$

Typing relation

Typing contexts: $\Gamma \quad ::= \quad \cdot \mid \Gamma, x :_r \tau$ for $r \in \mathbb{R}^{>0} \cup \{\infty\}$

"e is a well-formed expression of type τ in a context Γ:

$$\Gamma \vdash e : \tau$$

A c-sensitive function of x:

$$x :_c \tau_1 \vdash e : \tau_2$$

More generally:

$$x_1 :_{r_1} \tau_1, \ldots, x_n :_{r_n} \tau_n \vdash e : \tau$$

Metric preservation

Theorem Suppose $\Gamma \vdash e : \tau$. Let sequences of values $(v_i)_{1 \leq i \leq n}$ and $(v'_i)_{1 \leq i \leq n}$ be given. Suppose for all $i \in 1, \ldots, n$ that we have

1. $\vdash v_i, v'_i : \tau_i$
2. $d_{\tau_i}(v_i, v'_i) = s_i$
3. $x_i :_{r_i} \tau_i \in \Gamma$.

If the program $[v_1/x_1] \cdots [v_n/x_n]e$ evaluates to v, then there exists a v' such that $[v'_1/x_1] \cdots [v'_n/x_n]e$ evaluates to v', and

$$d_\tau(v, v') \leq \sum_i r_i s_i.$$

Talk to the authors this week!

DFuzz

Gaboardi, Haeberlen, Hsu, Narayan, and Pierce, *Linear Dependent Types for Differential Privacy,* POPL 2013

Motivation

- Primitives of Fuzz are similar to PINQ in expressiveness
  - But many useful programs are not understood by the typechecker
- Main shortcoming: Cannot track data-dependent function sensitivity

$$\mathit{2iter\text{-}k\text{-}means} :!_{\infty}\mathbf{L}(\mathbb{R}^2) \multimap !_{6\epsilon}\mathbb{R}^2\ \mathbf{set} \multimap \bigcirc(\mathbf{L}(\mathbb{R}^2))$$

Plan

- Enrich type system of Fuzz with *indexed types* capable of tracking such data dependencies

$$k\text{-}means : \forall i.(!_\infty \mathbf{N}[i] \multimap \mathbf{L}(\mathbb{R}^2)[k] \rightarrow !_{3i\epsilon}\mathbb{R}^2\ \mathbf{set} \multimap \bigcirc(\mathbf{L}(\mathbb{R}^2)[k]))$$

- Ongoing work
  - Paper to appear in POPL 2013
  - Prototype implementation underway
- Main challenge: Constraint solving

```
function kmeans
  (iters : Nat[i]) (eps : num[e])
  (db :[3 * i * e] (num, num) bag)
  (centers : list(num, num)[j])
  (iterate : num[e] -> (num, num) bag -o[3*e]
    list(num, num)[j] -> Circle list(num, num)[j])
  : Circle list(num, num)[j] {
  case iters of
    0      => return centers
  | n + 1 => sample next_centers =
      kmeans n eps db centers iterate;
      iterate eps db next_centers
  }
```

Status

- Prototype implementation underway
- Main issue: <u>constraint solving</u>
  - (hopefully using an SMT solver such as Z3)

Machine-Checked Proofs

Limitations (of language-based approaches)

- Each of the above approaches offers a fairly limited set of primitive datatypes (lists, bags, ...) and differentially private operations over them
  - The "reasons why" an algorithm is DP must be fairly straightforward
- Meanwhile, the algorithms community is continually generating clever new DP algorithms (often over other forms of data, e.g. graphs, streams)...

Possible approaches

- Add new primitives
- Drop the demand that privacy proofs be generated automatically
  - this leads to...

CertiPriv

Barthe, Köpf, Olmedo, Béguelin, Probabilistic Relational Reasoning for Differential Privacy, POPL 2012

,

CertiPriv

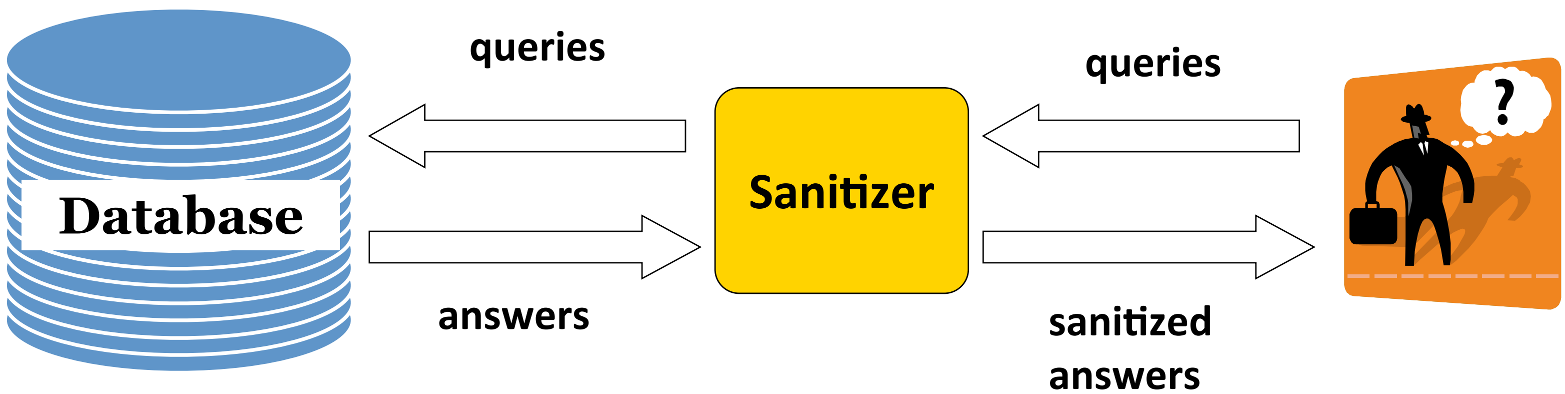
- Allows reasoning about approximate quantitative properties of randomized computations
  - Built from first principles and fully formalized in Coq
  - Machine-checked proofs of differential privacy
    - Correctness of Laplacian and Exponential mechanisms
    - State-of-art graph and streaming algorithms

DP for Interactive Systems

Tschantz, Kaynar, Datta, *Formal Verification of Differential Privacy for Interactive Systems*, MFPS 2011.

(Thanks to Anupam Datta for slides!)

Interactive model



System Model

- Bounded Memory
  - Cannot represent real numbers
  - Need discrete versions of privacy mechanisms
- Interactive I/O with environment
  - Answers queries over time
  - Also receiving new data points
- Probabilistic

Interaction Model

- Interleaving of data points, queries, and responses
- Mutable set of data points
- Adversary sees interleaving of queries and responses
- *Differential noninterference* generalizes both classical DP and classical noninterference from information-flow security
- Related to Pan Privacy [Dwork, Naor, Pitassi, Rothblum]
  - Maintains privacy for interactive systems under continual observation, even when the system's internal state is observed

Proof Technique

- Use local properties to imply the global property of differential privacy
- Use a refinement lemma to relate abstract models to concrete implementations
- Decompose verification into two problems:
  - Prove that sanitization functions have differential privacy using absorbing Markov chains
  - Prove that system correctly store data points and use sanitization functions using *unwinding*
- Partially automated

Some other work in PL

Continuity of Programs

- Observes that many everyday programs are
  - continuous (i.e., arbitrarily small changes to their inputs only cause arbitrarily small changes to their outputs)
  - or Lipschitz continuous (i.e., when their inputs change, their outputs change at most proportionally).
- Proposes a mostly-automatic framework for verifying that a program is continuous or Lipschitz

Chaudhuri, Gulwani, and Lublinerman. *Continuity analysis of programs*. POPL 2010
Chaudhuri, Gulwani, and Lublinerman. *Continuity and robustness of programs*. CACM, 2012.

DP in Process Calculi

- Consider a probabilistic process calculus as a specification formalism for concurrent systems
    - Framework for reasoning about differential privacy in this setting
- Illustrate ideas on an anonymity-preservation property for (an extension of) the Crowds protocol

Xu, *Modular Reasoning about Differential Privacy in a Probabilistic Process Calculus*, manuscript 2012

(Some!) Related Work in Systems

Talk to the authors this week!

Covert Channels

Haeberlen, Pierce, and Narayan, *Differential Privacy Under Fire*. USENIX Security, 2011

(Thanks to Andreas Haeberlen for slides!)

Covert-channel attacks

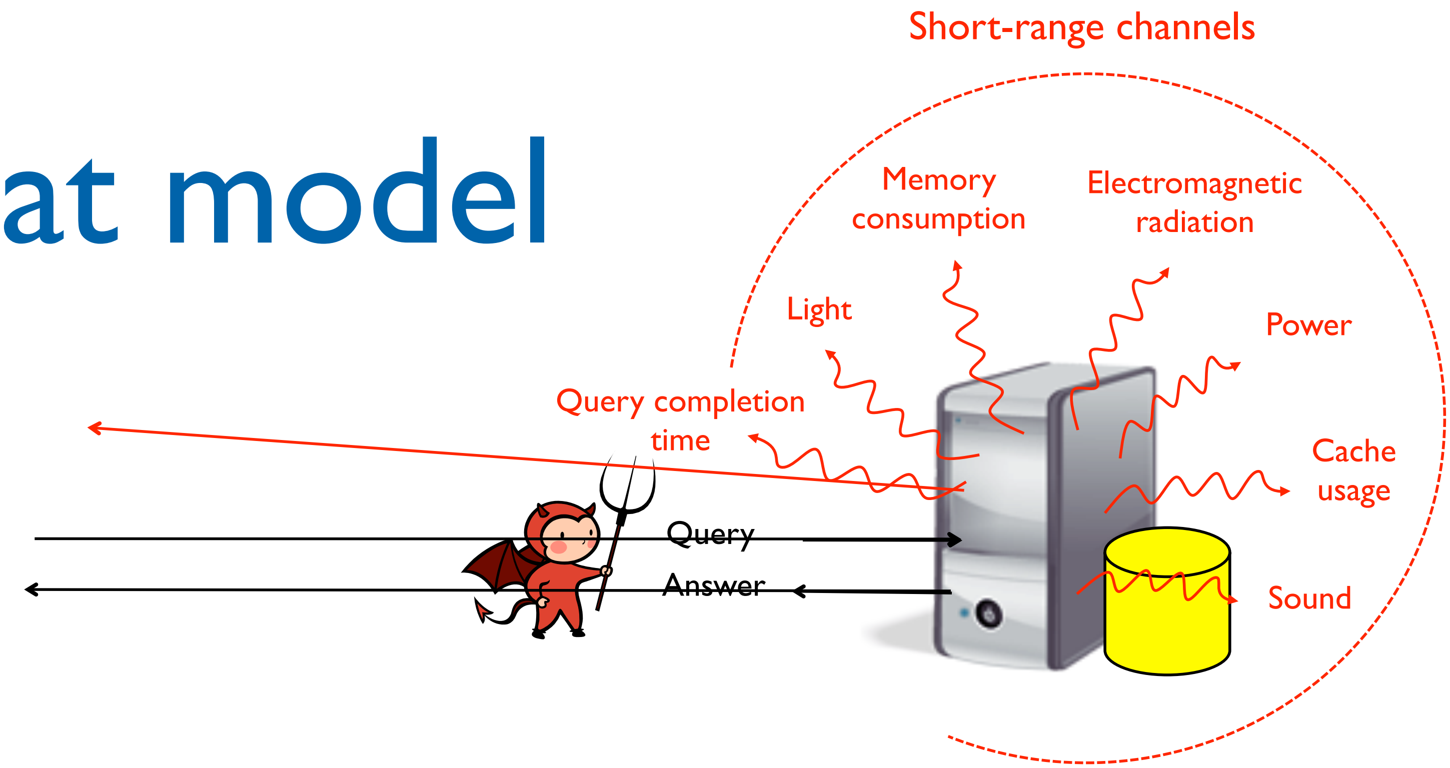
```
noisy sum, foreach r in db, of {
  if (r.name=="Bob" && r.hasRating("Porn"))
    then {
      b=1;
    };
  return b
}
```

- The above query...
  - ... is differentially private (sensitivity zero!)
  - ... takes 1 second longer if the database contains Bob's data
  - Result: Adversary learns private information with certainty!
- Other channels that can be exploited:
  - Global state
  - Privacy budget (!)

The attacks work in practice

- Both PINQ and Airavat are vulnerable
- What went wrong?
  - The authors were aware of this attack vector
  - Both papers discuss some ideas for possible defenses
  - But neither system has a defense that is fully effective

Threat model



- Too many channels!! Is it hopeless?
- Reasonable assumption: Querier is remote
- This leaves just two channels:
  - The actual answer to the query
  - The time until the answer arrives

Approach

- Close the remaining channels completely through a combination of systems and PL techniques
- Language design rules out state attacks etc.
  - Example: Simply don't allow global variables!
- Special runtime to close the timing channel

Plugging the timing channel

- How to avoid leaking information via query completion time?
  - Could treat time as an additional output
  - But: Unclear how to determine sensitivity

- Approach: Make timing predictable
  - If time does not depend on the contents of the database, it cannot leak information

Timeouts and default values

- Querier specifies for each “microquery”:
  - a timeout T, and
  - a default value d

- Each time the microquery processes a row:
  - If completed in less than T, wait
  - If not yet complete at T, abort and proceed to next row

Predictable transactions

- Isolation: Microquery must not interfere with the rest of the computation in any way
  - E.g. by triggering garbage collector, changing runtime state, ...
- Preemptability: Must be able to abort microqueries at any time
  - Even in the middle of memory allocation, ...
- Bounded deallocation: Must be able to free any allocated resources within bounded time
  - Example: Microquery allocates lots of memory, acquires locks...

- talk about going to epsilon-delta DP

$\boxed{\text{TODO}}$

Dangers of Floating Point

$$\texttt{float} \neq \mathbb{R}$$

- Duh...

See talk this week!

Ilya Mironov, *On Significance of the Least Significant Bits For Differential Privacy*, CCS 2012

Distributed DP

See talk this week!

Narayan and Haeberlen, *Differentially private join queries over distributed databases*, OSDI 2012

(Thanks to Andreas for slide!)

DJoin

- A differentially private query processor for distributed databases
- First practical solution that supports joins (with some restrictions)
- Based on two novel primitives
  - BN-PSI-CA: Differentially private set intersection cardinality
  - DCR: Denoise-combine-renoise
- Not fast enough for interactive use, but may be sufficient for offline data analysis

GUPT: Privacy Preserving Data Analysis Made Easy

Talk to authors this week!

Mohan, Thakurta, Shi, Song, and Culler. *GUPT: privacy preserving data analysis made easy*. SIGMOD 2012

GUPT: platform for differentially private execution of unmodified user code

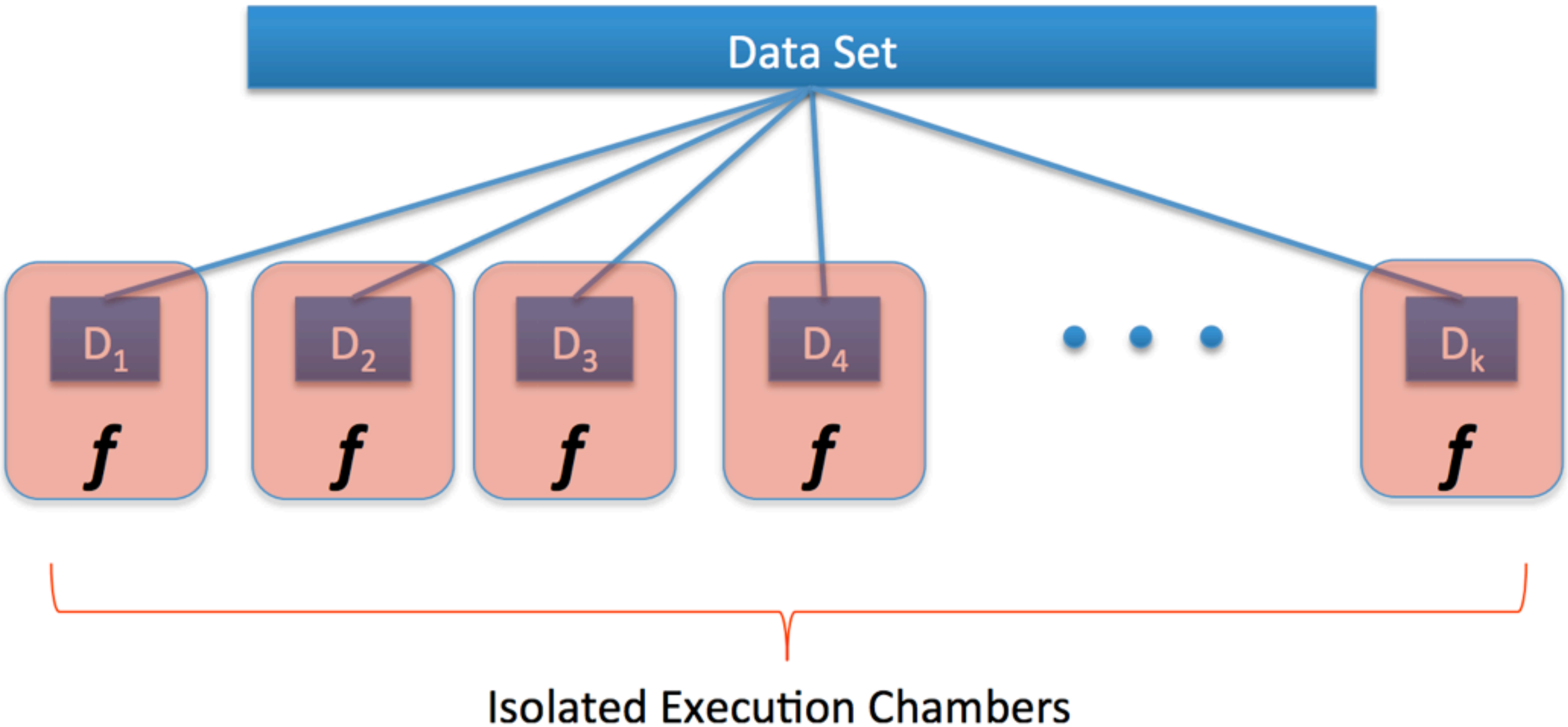
1. Improve output accuracy: resampling, optimal block size estimation

2. Usability: describing privacy budget in terms of accuracy, privacy budget allocation

3. Protection against side-channel attacks: state attack, privacy-budget attack, timing attack

Also: a new model of data sensitivity that degrades privacy of data over time. Enables efficient allocation of different levels of privacy for different applications while guaranteeing an overall constant level of privacy and maximizing utility

Main idea: Sample and Aggregate [NRS07, Smith11]



Winding Up...

Challenges

- Balancing expressiveness and automation
- Bullet-proof implementations
- Extending the tools with a broader range of data structures (graphs, streams) and DP algorithms

- Realistic examples!

Thank you!

Any questions?

http://privacy.cis.upenn.edu