Differential Privacy Under Fire

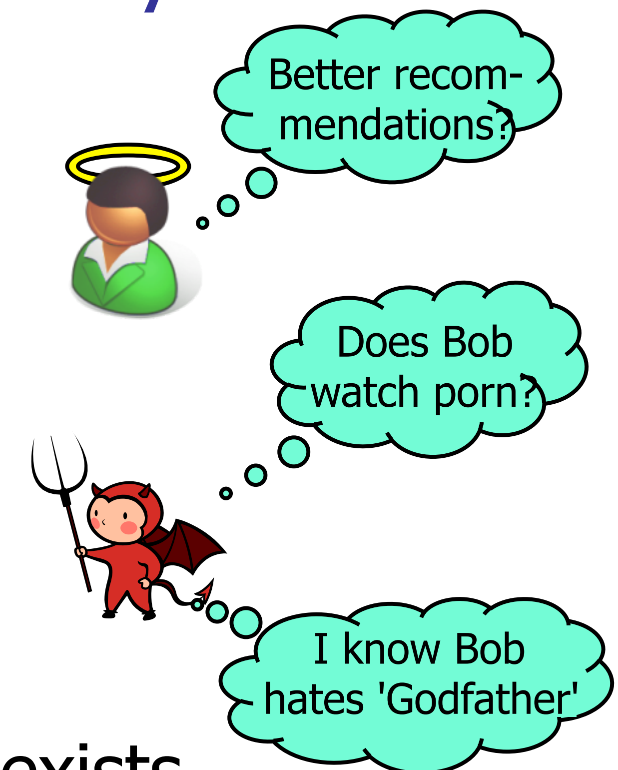
Andreas Haeberlen Benjamin C. Pierce Arjun Narayan

University of Pennsylvania

Motivation: Protecting privacy

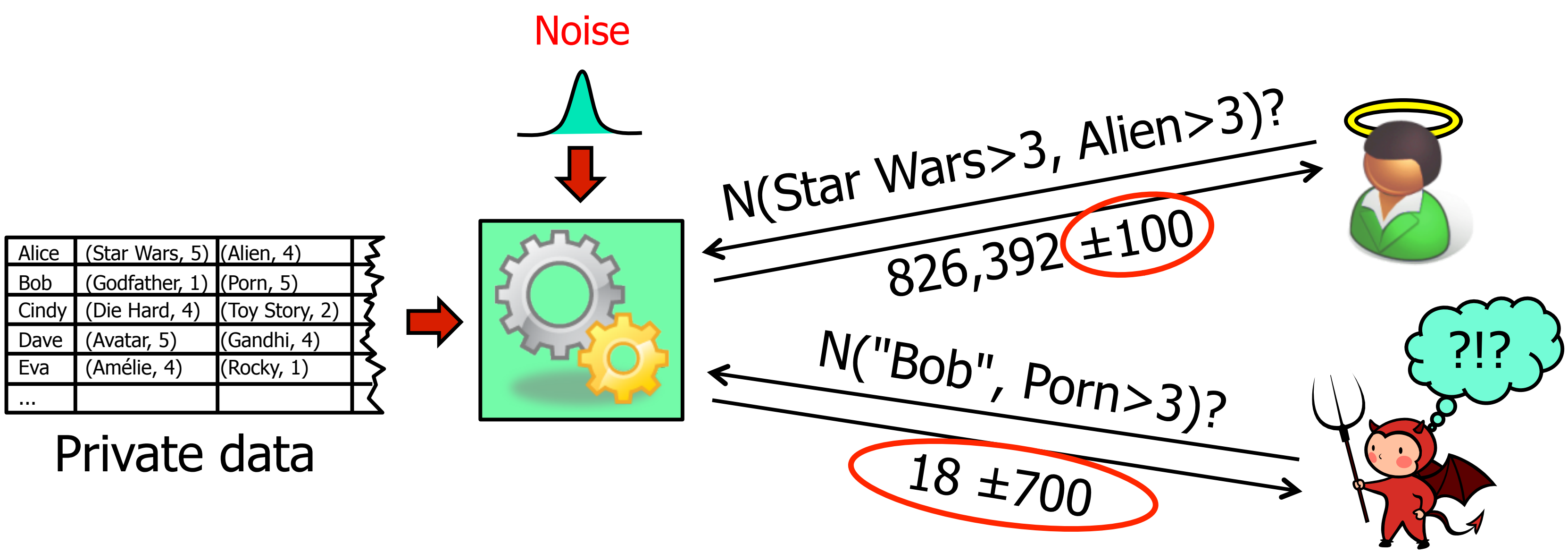
#1	(Star Wars, 5)	(Alien, 4)	
#2	(Godfather, 1)	(Porn, 5)	
#3	(Die Hard, 4)	(Toy Story, 2)	
#4	(Avatar, 5)	(Gandhi, 4)	
#5	(Amélie, 4)	(Rocky, 1)	
...			

Data



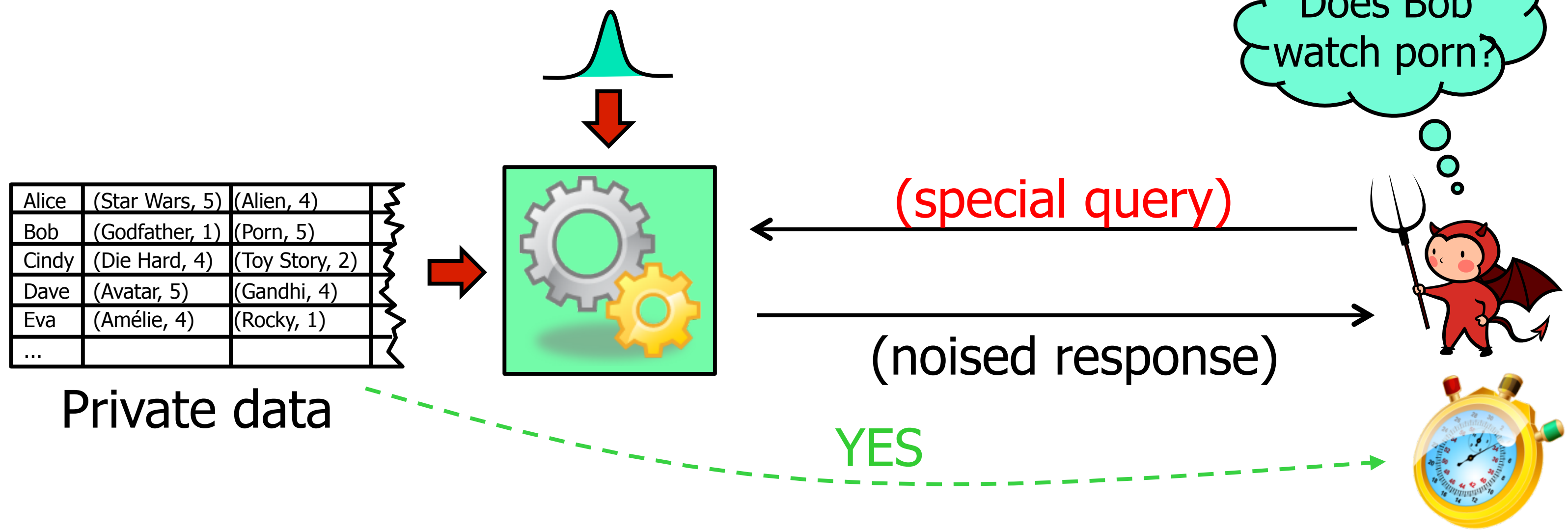
- Lots of potentially useful data exists
- But: Releasing it can violate privacy!
  - We can try to anonymize/scrub it…
  - … but this can go horribly wrong (see Netflix, AOL, …)

Promising approach: Differential privacy



- Idea: Use differential privacy [Dwork et al.]
  - Only allow queries; add a certain amount of noise to results
  - [lots of mathematical details omitted]
  - Result: Strong, provable privacy guarantees
  - Implemented, e.g., by PINQ [McSherry] and Airavat [Roy et al.]

Differential Privacy under Fire



Alice	(Star Wars, 5)	(Alien, 4)
Bob	(Godfather, 1)	(Porn, 5)
Cindy	(Die Hard, 4)	(Toy Story, 2)
Dave	(Avatar, 5)	(Gandhi, 4)
Eva	(Amélie, 4)	(Rocky, 1)
...		

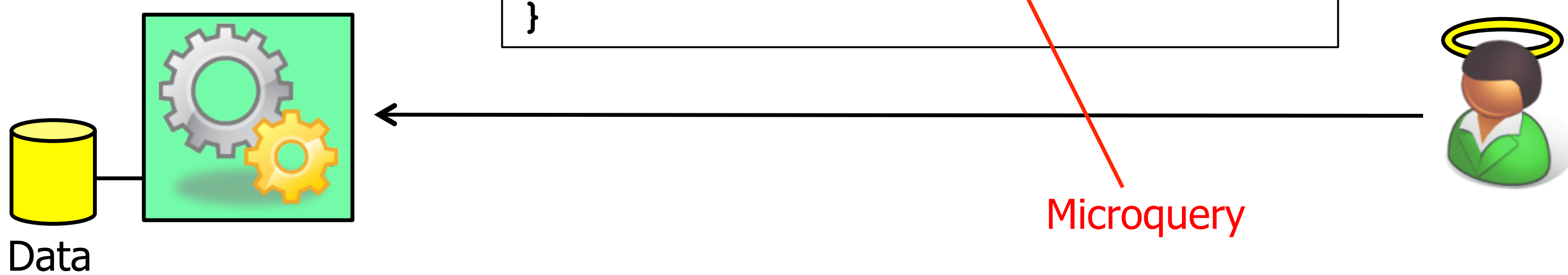
- What if the adversary uses a covert channel?
  - Devastating effect on privacy guarantees
  - Usual defenses are not strong enough (can't leak even one bit!)
- We show:
  - Working attacks
  - An effective (domain-specific) defense

Outline

- Motivation
- Differential Privacy primer NEXT
- Attacks on PINQ and Airavat
- Our defense
- The Fuzz system
- Evaluation

Background: Queries

```
noisy sum, foreach r in db, of {
  if (r.score("Godfather")>4)
    then return 1
    else return 0
}
```



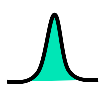
- Queries are programs
  - PINQ is based on C#, Airavat on MapReduce
- These programs have a specific structure
  - Some overall program logic, e.g., aggregation
  - Some computation on each database row (microquery)

Background: Sensitivity

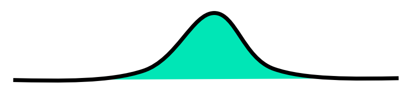
```
noisy sum, ∀r in db, of {
  if (r.score("Godfather")>4)
    then return 1
    else return 0
}
```

```
noisy sum, ∀r in db, of {
  if (r.score("Godfather")>4)
    then return 1200
    else return 200
}
```

Sensitivity 1

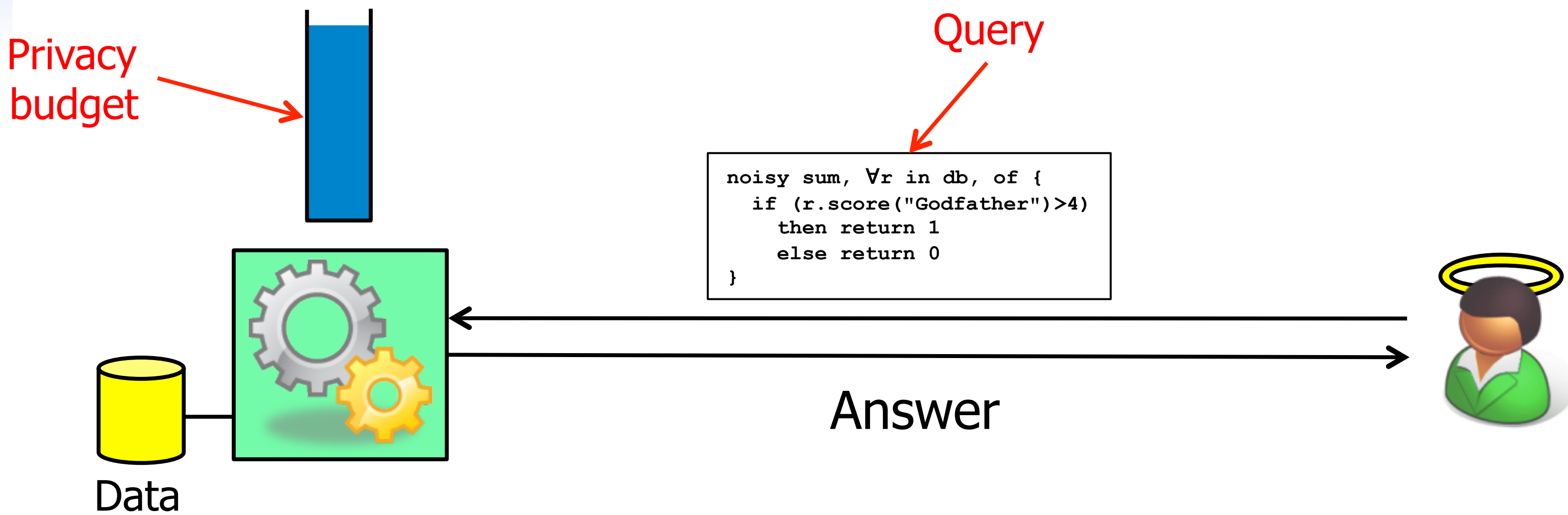


Sensitivity 1,000



- How much noise should we add to results?
  - Depends on how much the output can change if we add or remove a single row (the sensitivity of the query)

Background: Privacy budget



- How many queries should we answer?
  - Set up a privacy 'budget' for answering queries
  - Deduct a 'cost' for each query, depending on 'how private' it is

Covert-channel attacks

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

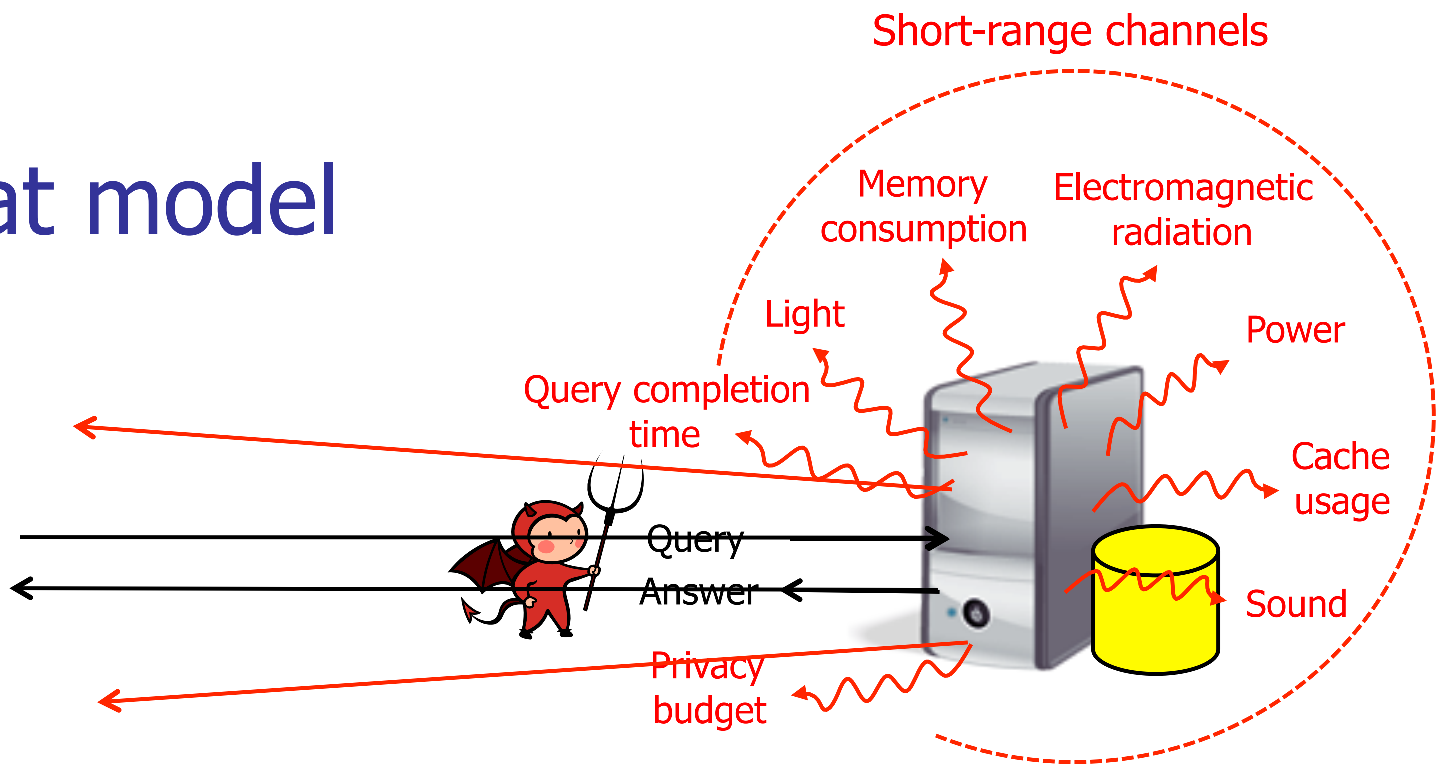
- The above query...
  - ... is differentially private (sensitivity zero)
  - ... takes 1 second longer if the database contains Bob's data
  - Result: Adversary can learn private information with certainty!
- Other channels we have exploited:
  - Privacy budget
  - Global state

Our 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 three channels:
  - The actual answer to the query
  - The time until the answer arrives
  - The decision whether the remaining budget is sufficient

Our approach

- We can 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!

- Program analysis closes the budget channel
  - Idea: Statically determine the 'cost' of a query before running it
  - Uses a novel type system [Reed and Pierce]

Details in the paper

- Special runtime to close the timing channel

NEXT

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

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

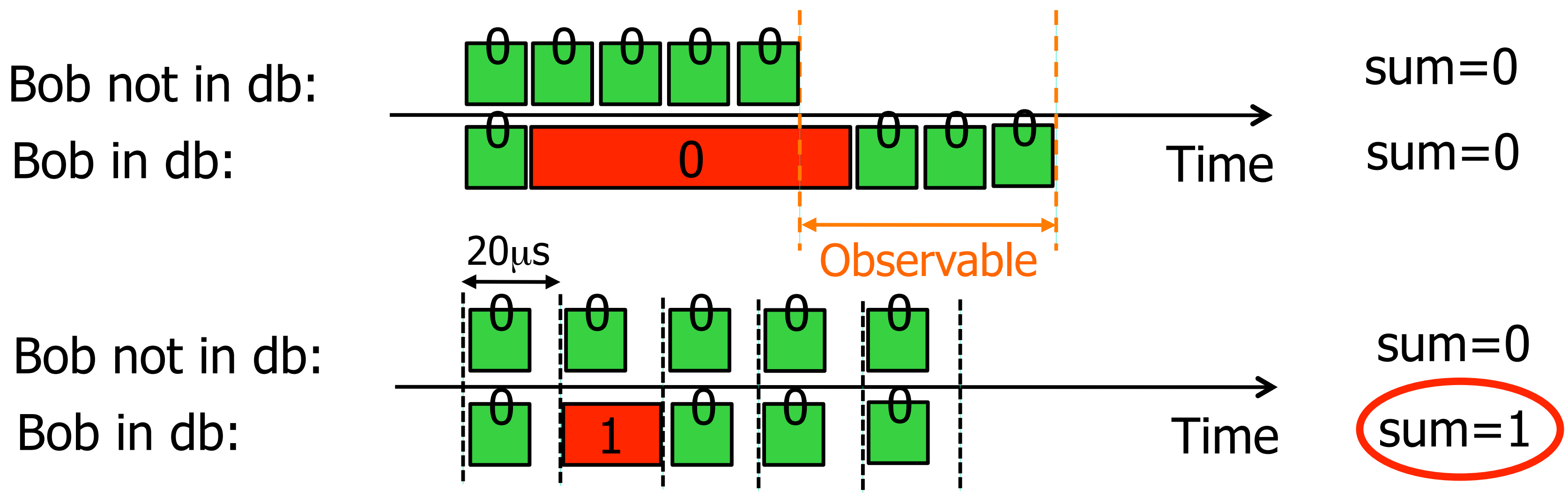
Timeouts and default values

- 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

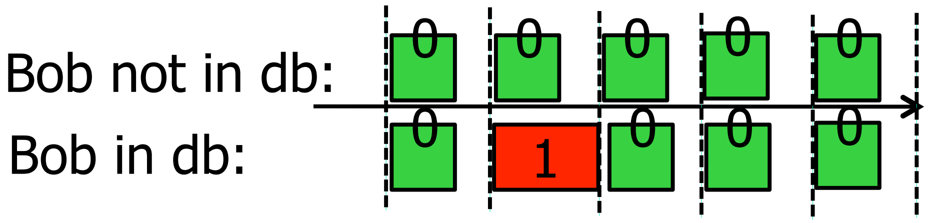
Example: Timeouts and default values

Alice	(Star Wars, 5)	(Alien, 4)	
Rob	(Godfather, 1)	(Porn, 5)	
Cindy	(Die Hard, 4)	(Toy Story, 2)	
Dave	(Avatar, 5)	(Gandhi, 4)	
Eva	(Amélie, 4)	(Rocky, 1)	

```
noisy sum, ∀r∈db, of {
  if r.name=="Bob"
    then loop(1 sec);
  return 0
}, T=20μs, d=1
```



Default values do not violate privacy




```
noisy sum, ∀r∈db, of {
  if r.name=="Bob"
    then loop(1 sec);
  return 0
}, T=20μs, d=1
```

- Don't default values change the query's answer?
- Yes, but that's okay:
  - Remember that the answer is still noised before it is returned
  - Noise depends on the sensitivity, which is now 1
  - It's just as if we had written "If r.name=='Bob', return 1"
- Impact on non-adversarial queriers?
  - Default value is never included if timeout is sufficiently high

Outline

- Motivation ✓
- Differential Privacy primer ✓
- Attacks on PINQ and Airavat ✓
- Our defense ✓
- The Fuzz system ← NEXT
- Evaluation

The Fuzz system

- Fuzz: A programming language for writing differentially private queries
  - Designed from scratch → Easier to secure
  - Functionality roughly comparable to PINQ/Airavat
  - Novel type system for statically checking sensitivity
- Runtime supports timeouts + default values
  - Turns out to be highly nontrivial
  - Problem: How to make a potentially adversarial computation take exactly a given amount of time?
  - Uses a new primitive called predictable transactions

Predictable transactions

- Isolation: Microquery must not interfere with the rest of the computation in any way
  - Examples: Trigger garbage collector, change 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...
- Details are in the paper

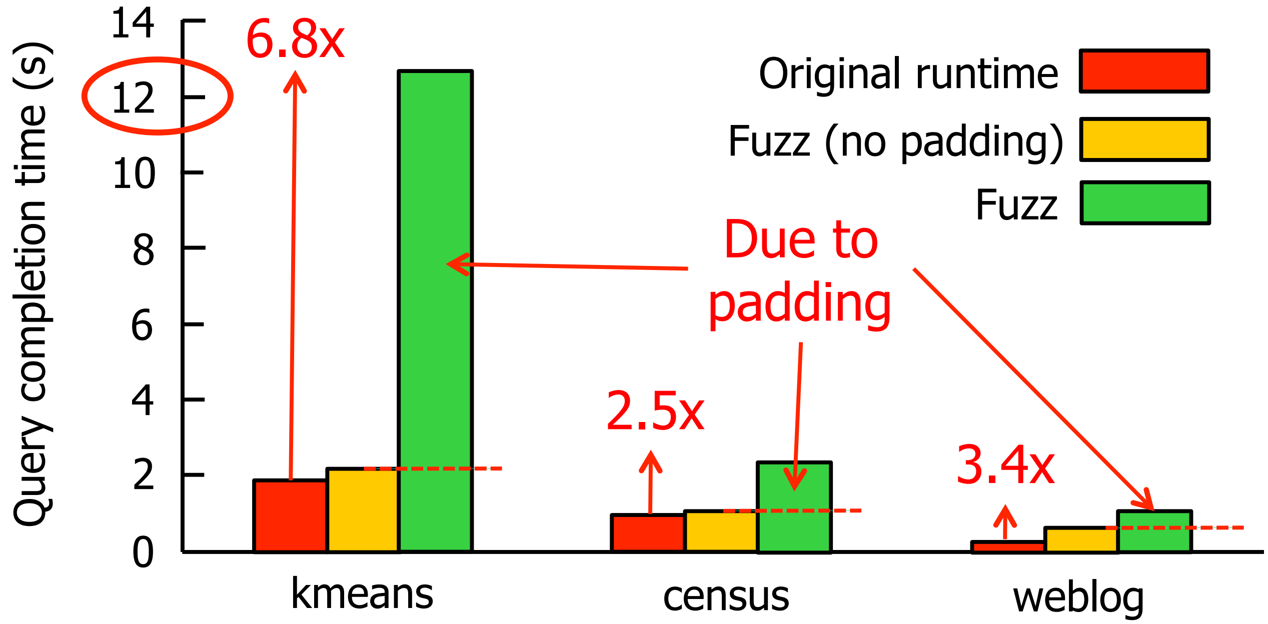
Outline

- Motivation ✓
- Differential Privacy primer ✓
- Attacks on Differential Privacy ✓
- Defenses ✓
- The Fuzz system ✓
- Evaluation ← NEXT
  - Is Fuzz expressive enough to handle realistic queries?
  - Is Fuzz fast enough to be practical?
  - Does Fuzz effectively prevent side-channel attacks?
  - More experiments are described in the paper

Experimental setup

- Implemented three queries from prior work:
  - K-means clustering (inspired by Blum et al., PODS'05)
  - Census query (inspired by Chawla et al., TCC'05)
  - Web server log analysis (inspired by Dwork et al., TCC'06)
  - Fuzz is expressive enough to run all three queries
- Also crafted several adversarial queries
  - Using different variants of our attacks
- Evaluated on a commodity system
  - 3GHz Core 2 Duo running Linux 2.6.38
  - Synthetic database with 10,000 rows

Performance: Non-adversarial queries



- Query completion time increased by 2.5x-6.8x
  - But: Most expensive query took 'only' 12.7s
- Most of the increase was due to time padding
  - Timeouts were set conservatively
  - More detailed results are in the paper

Performance: Adversarial queries

#	Attack type	Protection disabled Hit	Miss	Δ	Protected Hit	Miss	Δ
1	Memory allocation	1.96s	0.32s	1.6s	1.10s	1.10s	<1μs
2	Garbage collection	1.57s	0.32s	1.2s	1.10s	1.10s	<1μs
3	Artificial delay	1.62s	0.32s	1.3s	1.10s	1.10s	<1μs
4	Early termination	26.37s	26.38s	6ms	1.10s	1.10s	<1μs
5	Artificial delay	2.17s	0.90s	1.3s	2.40s	2.40s	<1μs

- Evaluated five adversarial queries
  - Unprotected runtime: Attacks cause large timing variation
  - Protected runtime: Completion times are extremely stable
- Timing channel now too narrow to be useful!
  - Remember: State and budget channels closed by design

Summary

- Differentially private query processors must be protected against covert-channel attacks
  - Leaking even a single bit can destroy the privacy guarantees
- Vulnerabilities exist in PINQ and Airavat
- Proposed defense: Fuzz
  - Uses static analysis and predictable transactions
  - Specific to differential privacy, but very strong: Closes all remotely measurable channels completely

More information at: **http://privacy.cis.upenn.edu/**