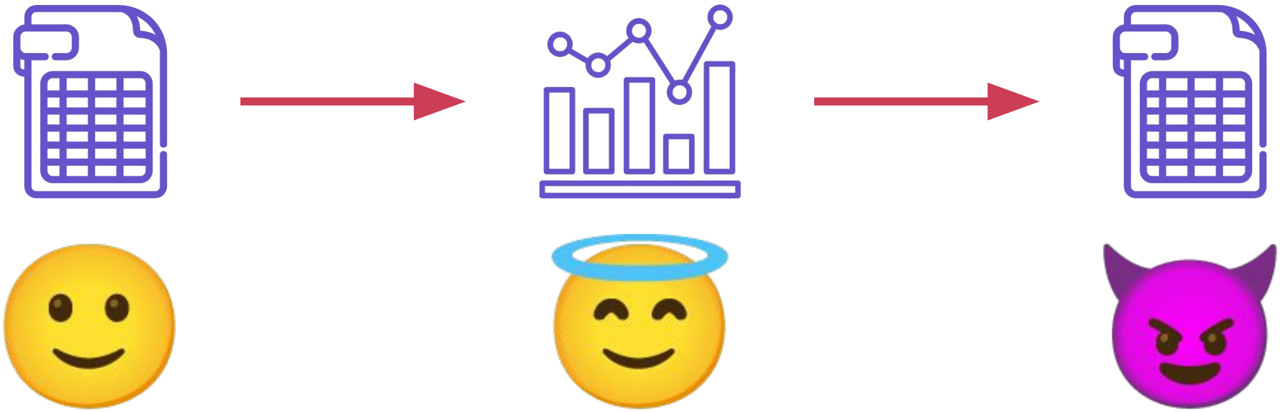
How to break, then fix, differential privacy on finite computers

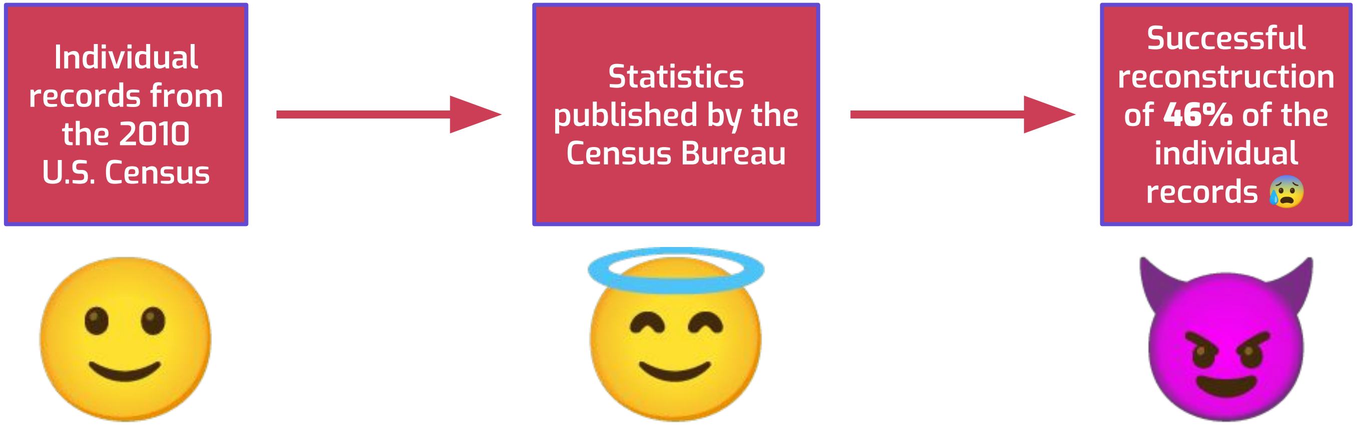
Or: what do you do when x + y = privacy vulnerability?

Damien Desfontaines
damien@desfontain.es
@TedTed@hachyderm.io

Background: the problem



Background: the problem



Individual records from the 2010 U.S. Census

🙂

Statistics published by the Census Bureau

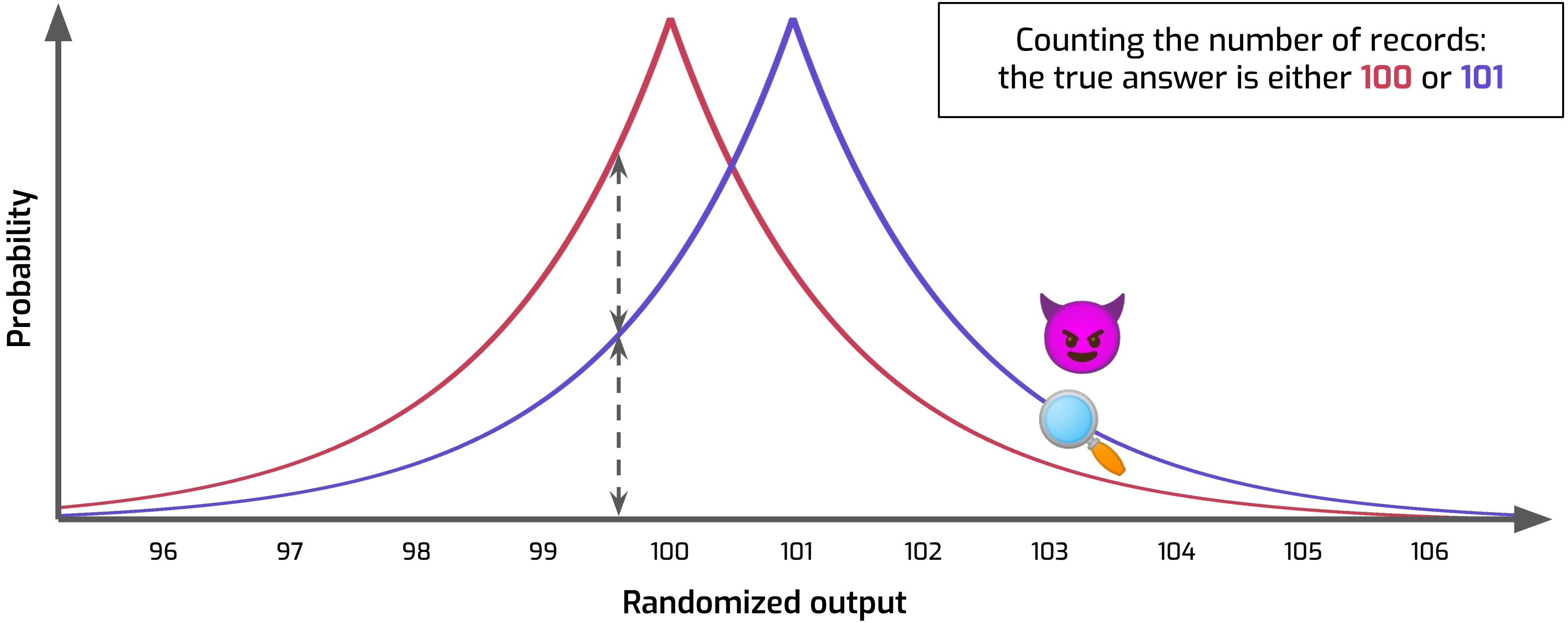
😇

Successful reconstruction of **46%** of the individual records 😰

😈

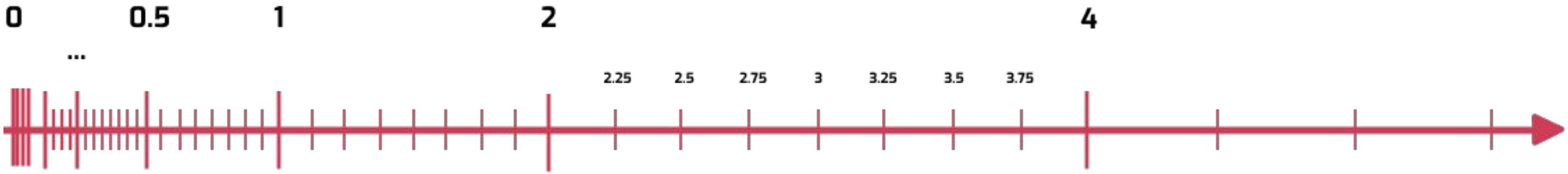
Background: the solution, in theory

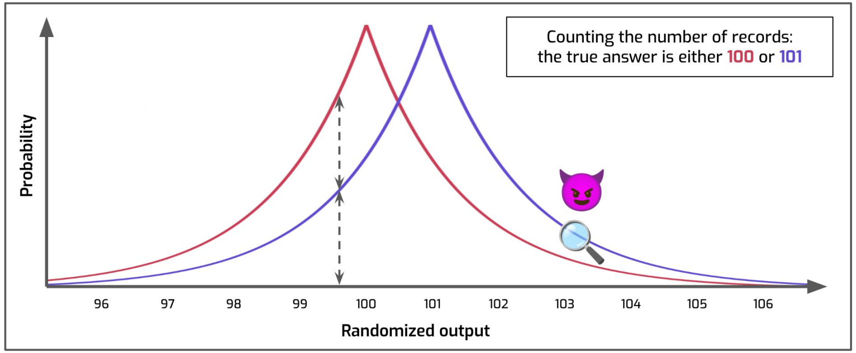
Differential privacy: the impact of a single person must be **undetectable**.



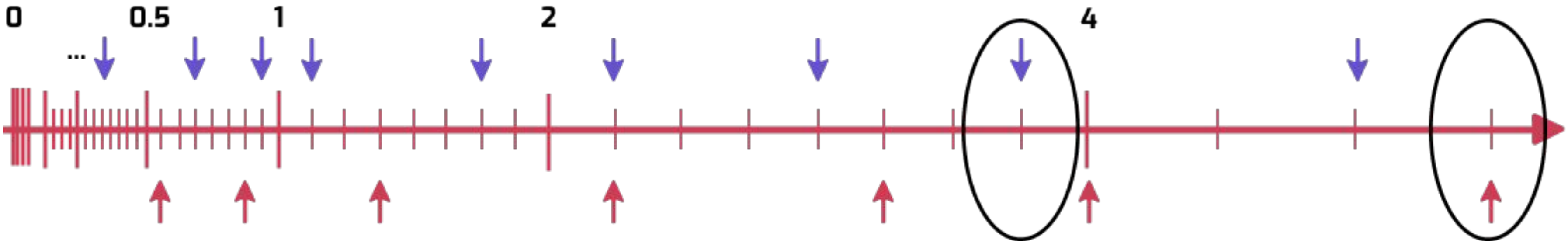
Counting the number of records:
the true answer is either 100 or 101

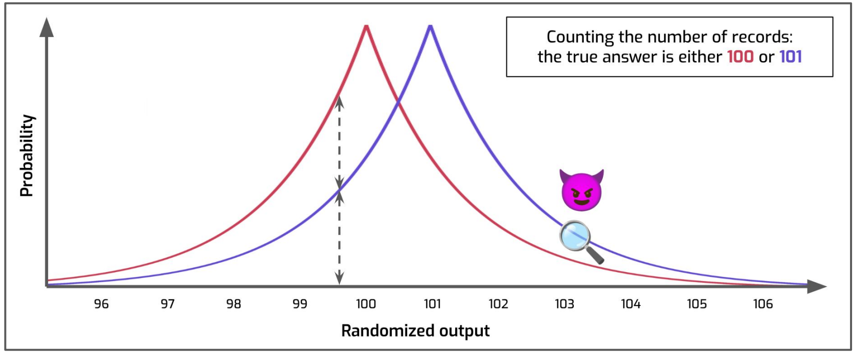
Zooming in: floating-point numbers




Counting the number of records:
the true answer is either 100 or 101

What happens to our continuous line?





Why does this happen?

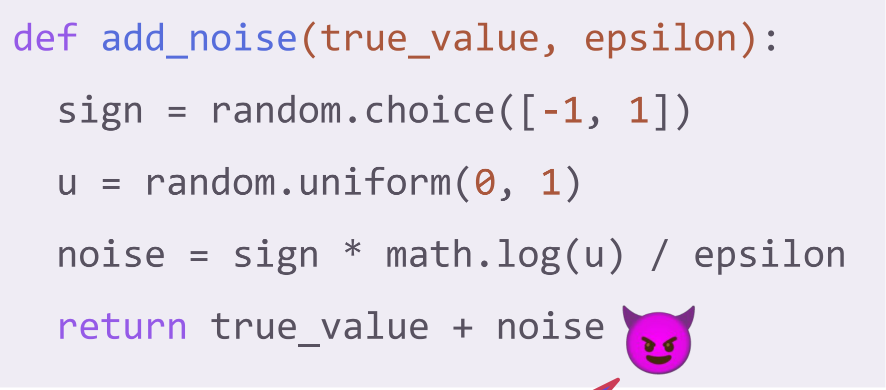
```python
def add_noise(true_value, epsilon):
  sign = random.choice([-1, 1])
  u = random.uniform(0, 1)
  noise = sign * math.log(u) / epsilon
  return true_value + noise
```

This does not generate all possible floating-point values between 0 and 1!

This creates “holes” — impossible values — in the noise distribution…

And the “holes” propagate to the sum.

Let’s fix the noise generation!



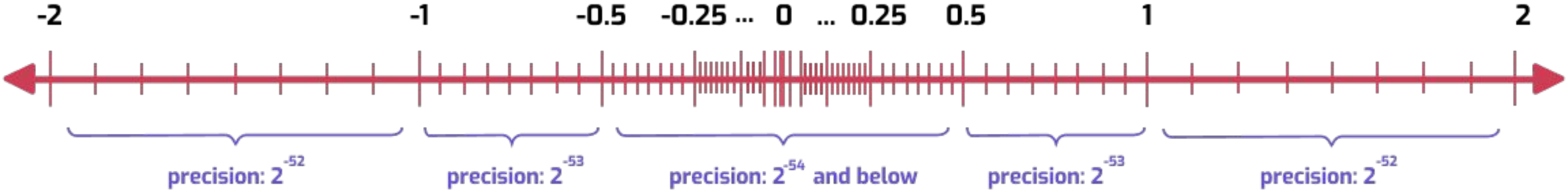
```
def add_noise(true_value, epsilon):
  sign = random.choice([-1, 1])
  u = random.uniform(0, 1)
  noise = sign * math.log(u) / epsilon
  return true_value + noise 😈
```

But… what about the sum at the very end?

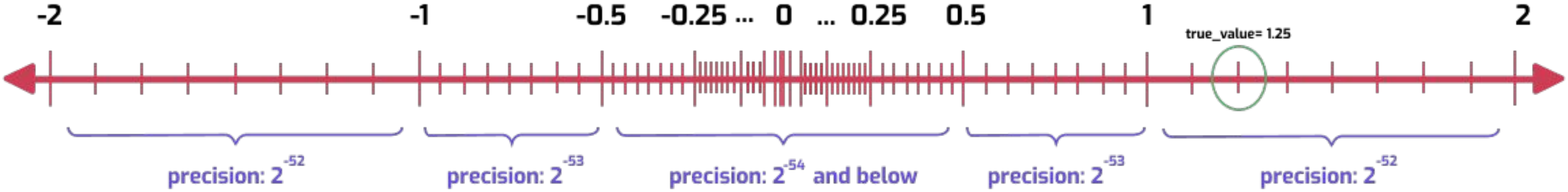
Attempt 1: fixing the noise generation to get a distribution without “holes”.

Attempt 2: combining multiple noise samples together to make it intractable to reverse-engineer the randomness.

Fun fact about floating-point addition…

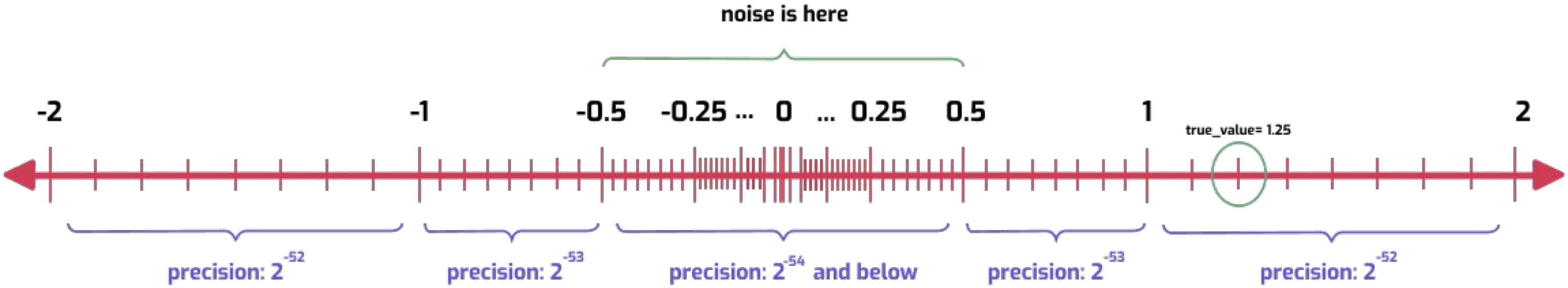


Fun fact about floating-point addition...



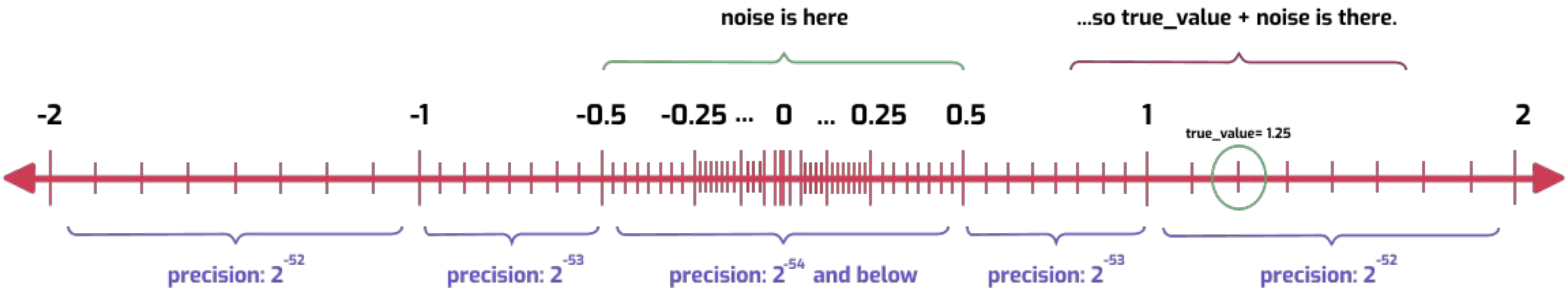
What if we add noise to 1.25?
It has precision 2^{-52}.

Fun fact about floating-point addition...



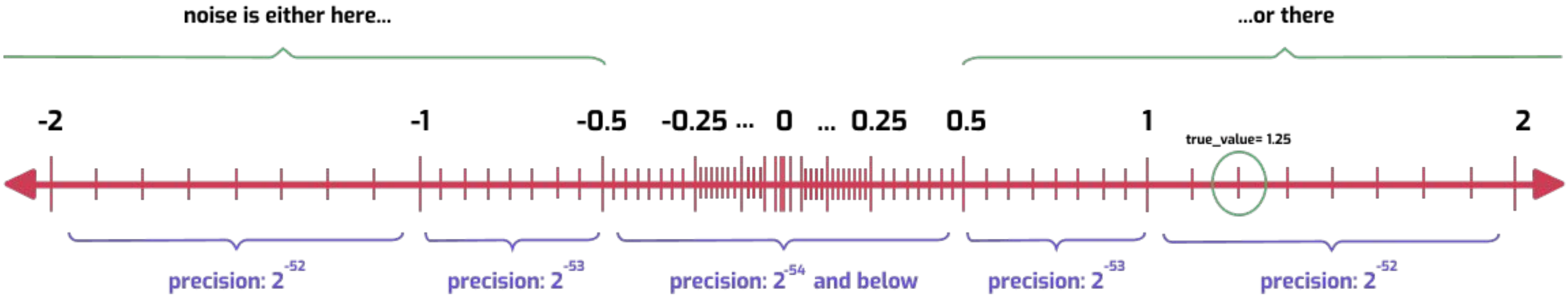
If the noise is small...

Fun fact about floating-point addition…



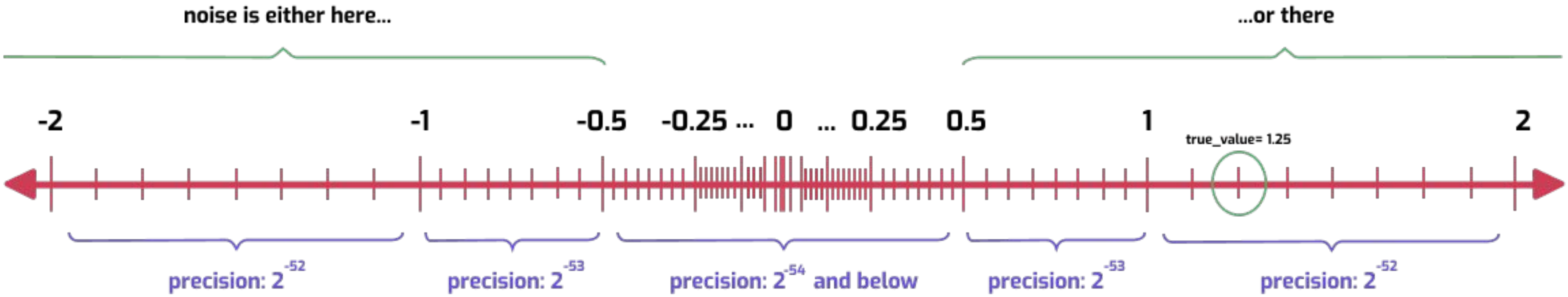
If the noise is **small**…
the sum’s precision is at least 2^{-53}.

Fun fact about floating-point addition…



If the noise is **large**…

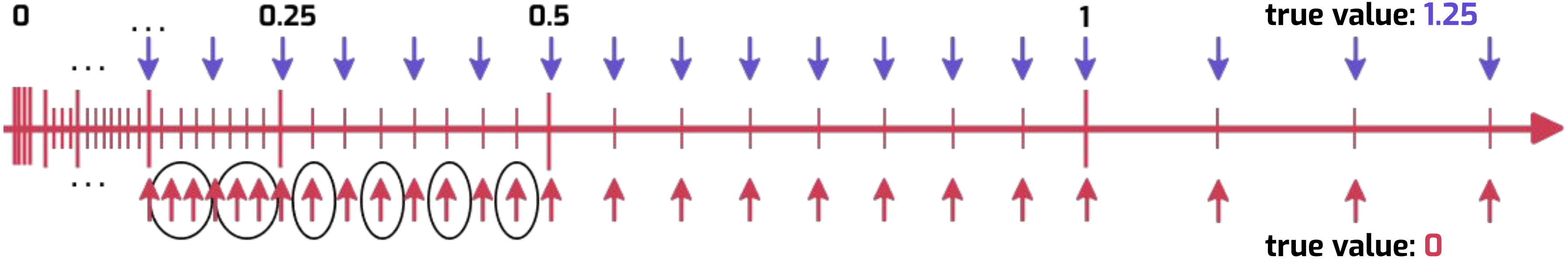
Fun fact about floating-point addition...



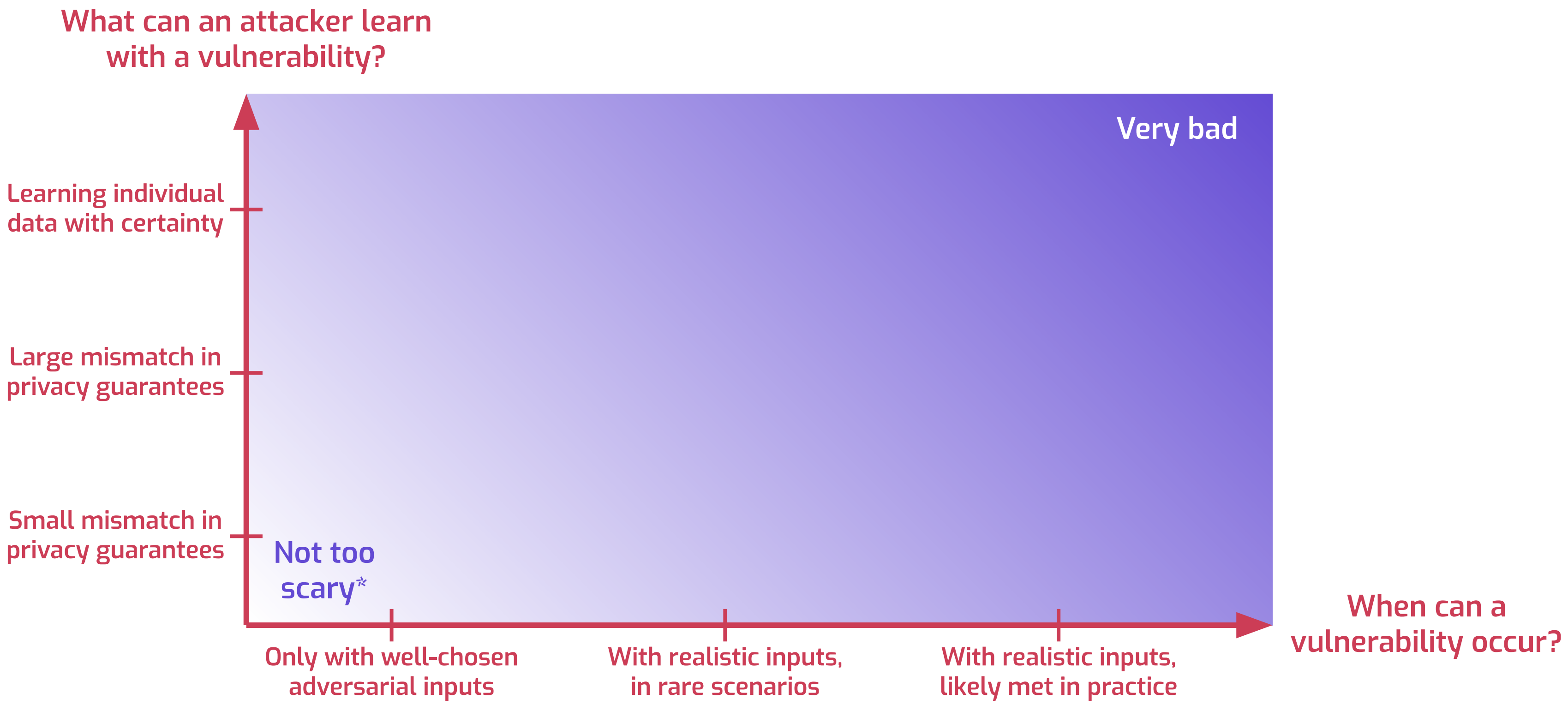
If the noise is **large**...
the sum is a multiple of 2^{-53}!

Takeaway: this is bad news

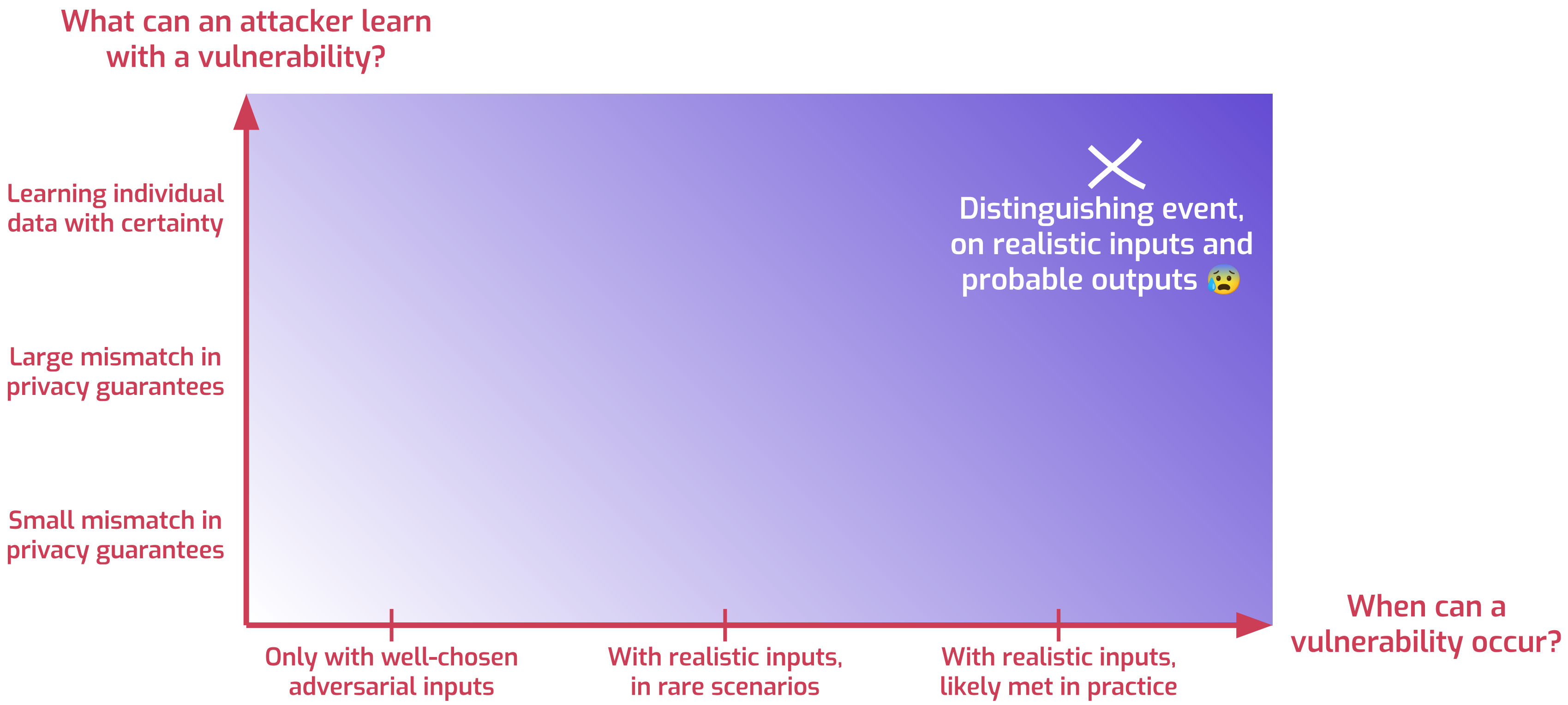
When adding noise to a number of precision 2^k, **we always get a multiple of 2^{k-1}**.



How bad is this?


What can an attacker learn with a vulnerability?
Learning individual data with certainty
Large mismatch in privacy guarantees
Small mismatch in privacy guarantees
Very bad
Not too scary*
Only with well-chosen adversarial inputs
With realistic inputs, in rare scenarios
With realistic inputs, likely met in practice
When can a vulnerability occur?

How bad is this?

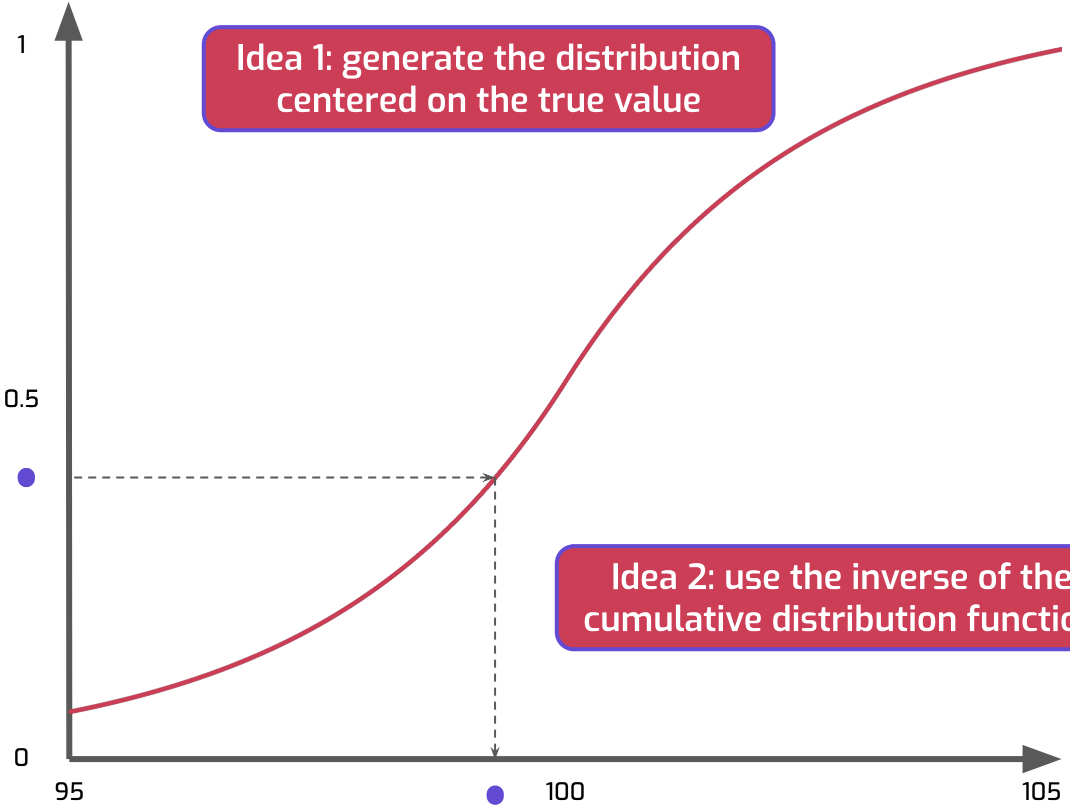

What can an attacker learn with a vulnerability?
Learning individual data with certainty
Large mismatch in privacy guarantees
Small mismatch in privacy guarantees
Distinguishing event, on realistic inputs and probable outputs 😰
When can a vulnerability occur?
Only with well-chosen adversarial inputs
With realistic inputs, in rare scenarios
With realistic inputs, likely met in practice

How do we fix it?

```
def add_noise(true_value, epsilon):
  sign = random.choice([-1, 1])
  u = random.uniform(0, 1)
  noise = sign * math.log(u) / epsilon
  return true_value + noise
```

We need to fix the entire routine, not just the noise generation!

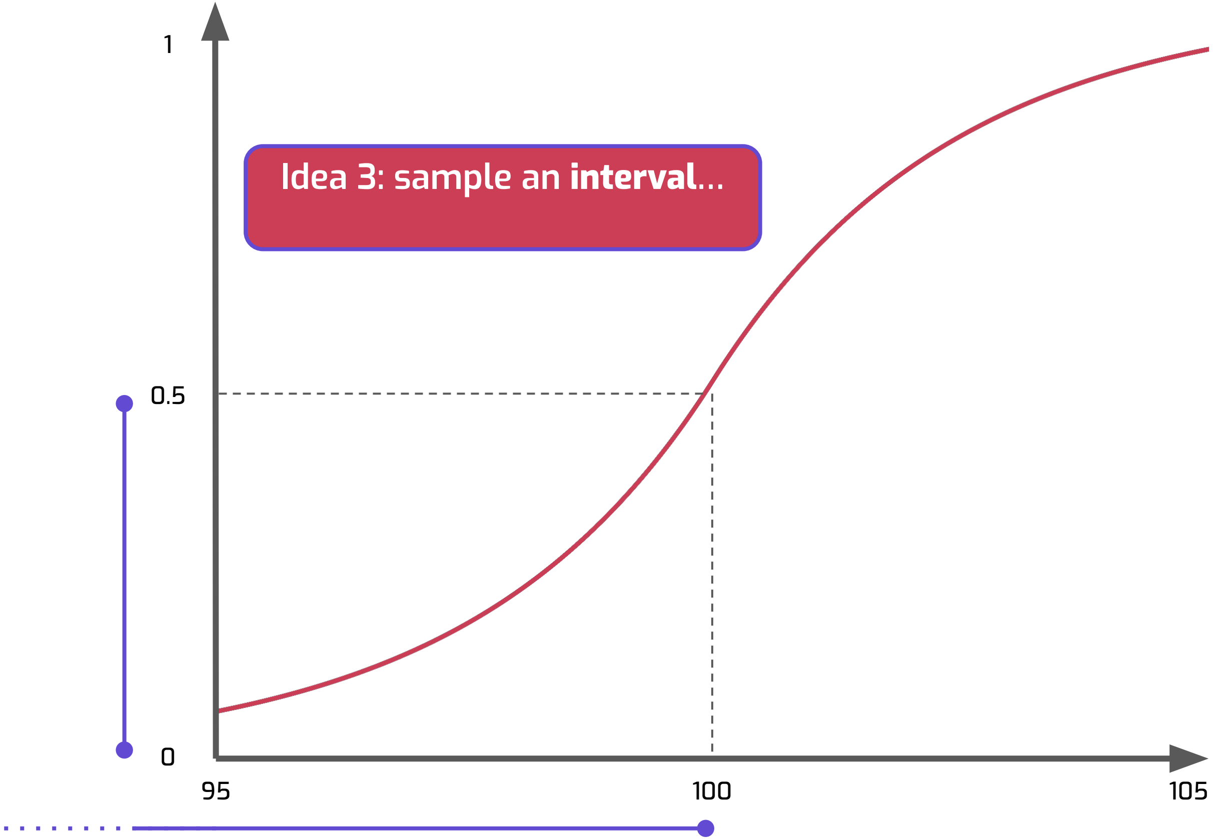
Four core ideas



Idea 1: generate the distribution centered on the true value

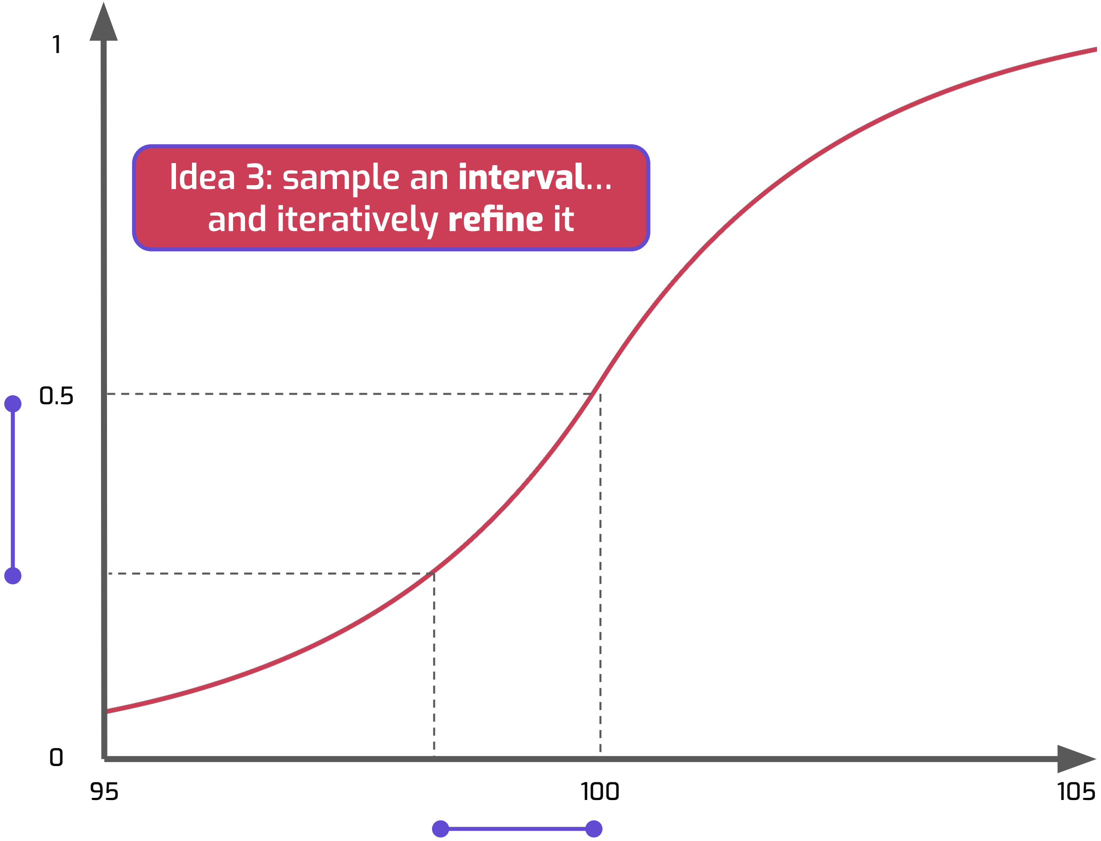
Idea 2: use the inverse of the cumulative distribution function

Four core ideas



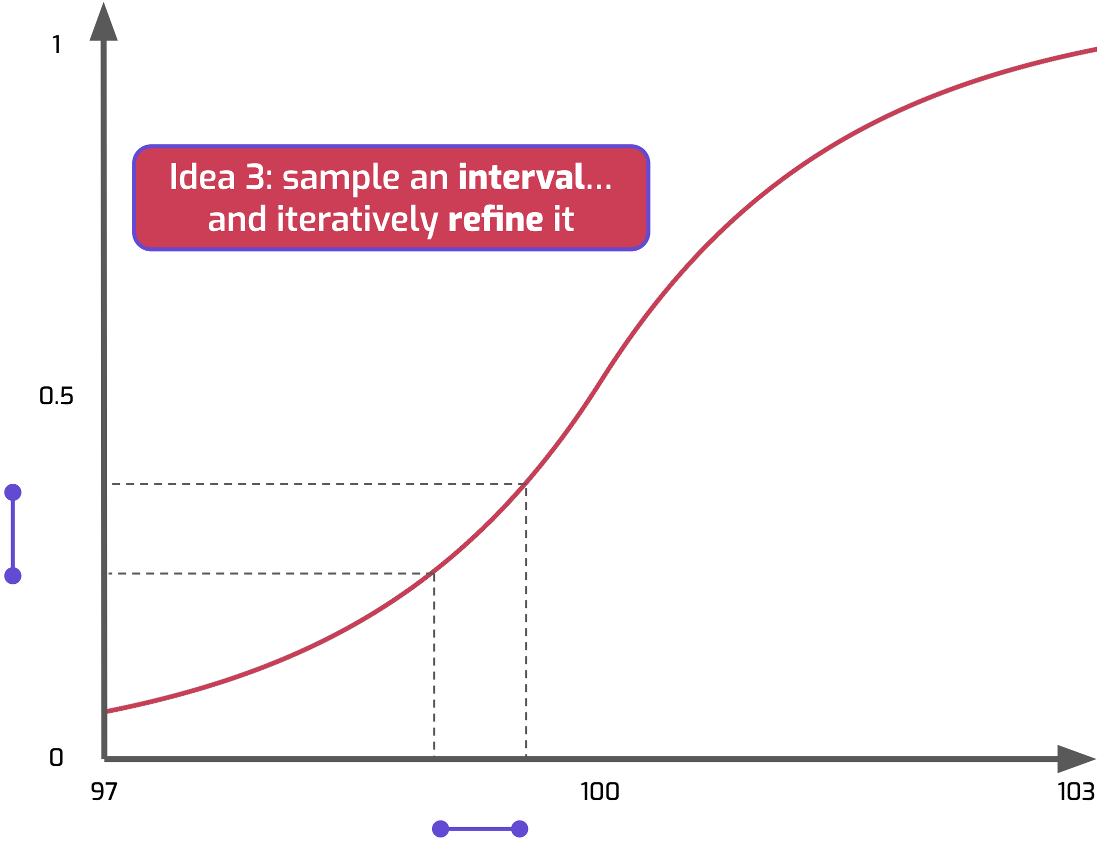
Idea 3: sample an **interval**…

Four core ideas



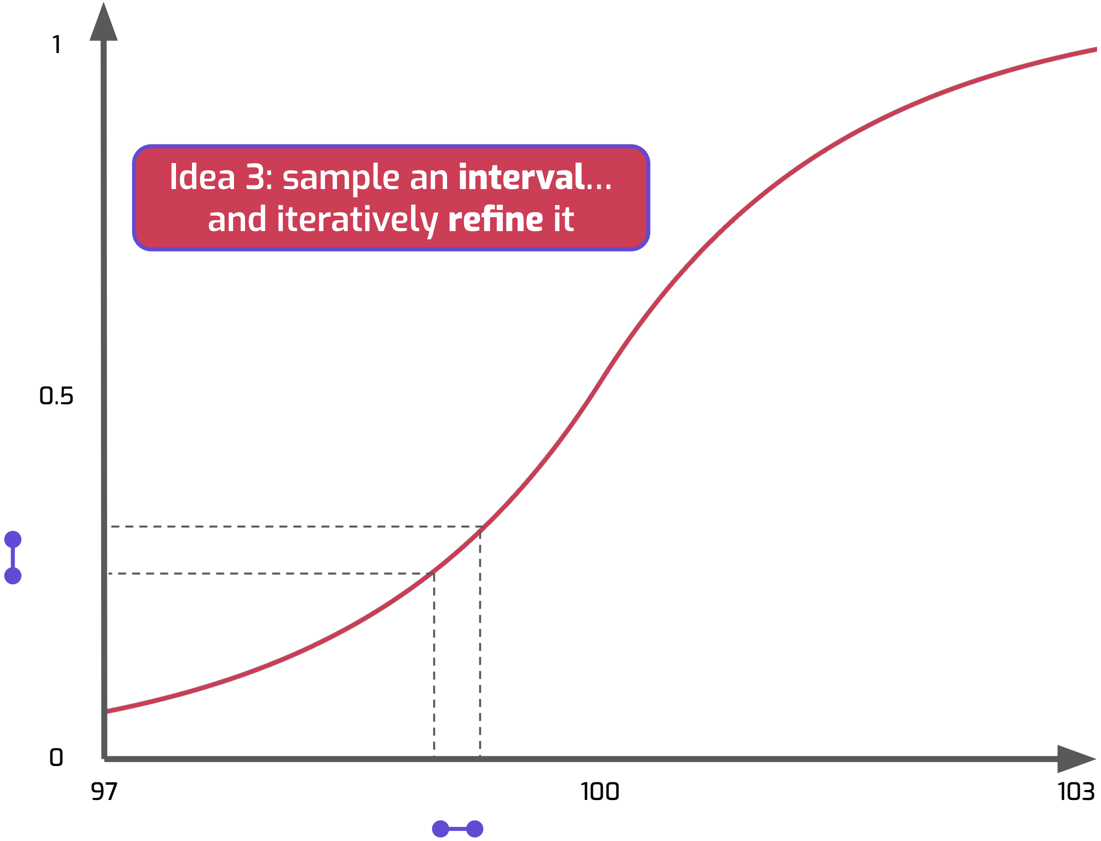
Idea 3: sample an **interval**... and iteratively **refine** it

Four core ideas



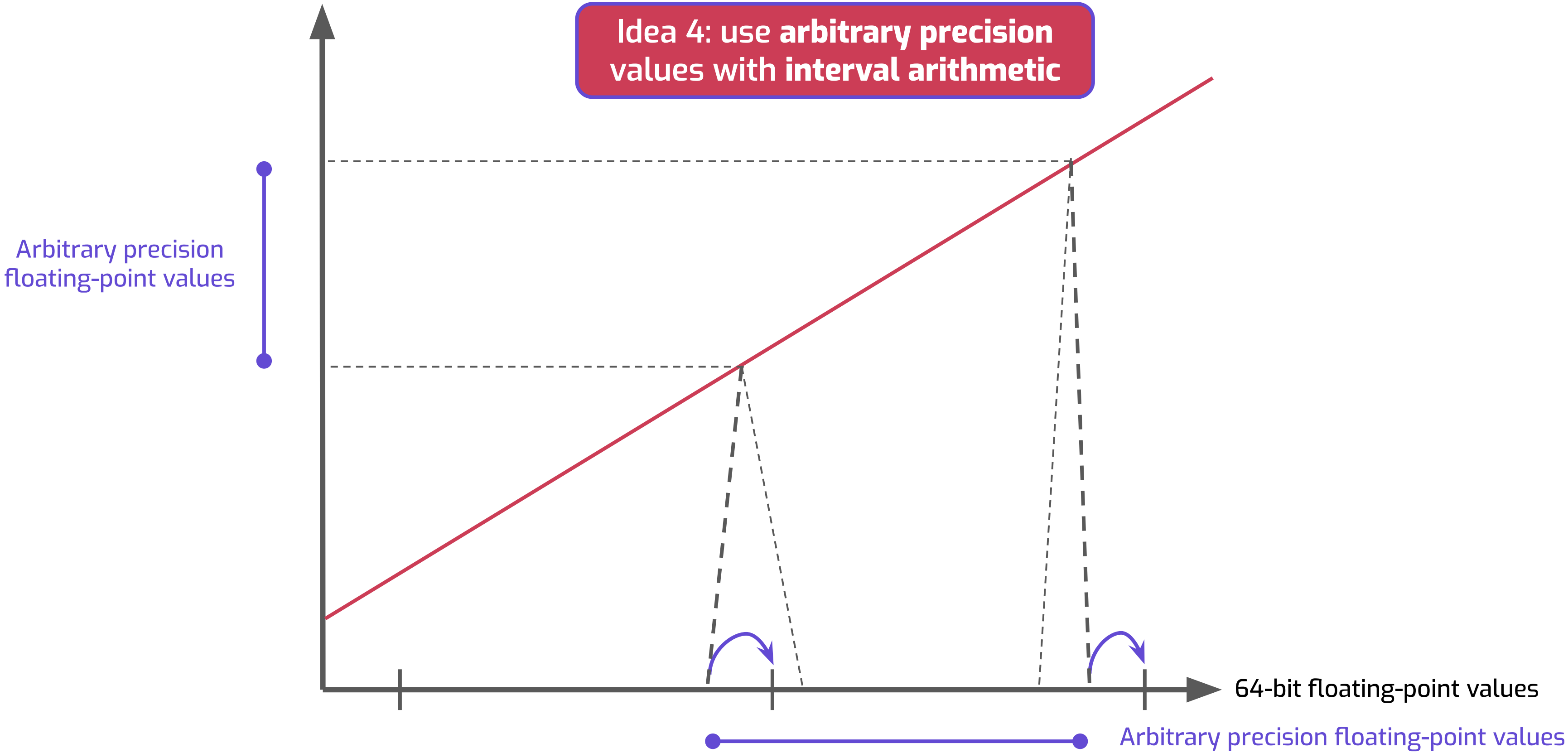
Idea 3: sample an **interval**…
and iteratively **refine** it

Four core ideas



Idea 3: sample an **interval**…
and iteratively **refine** it

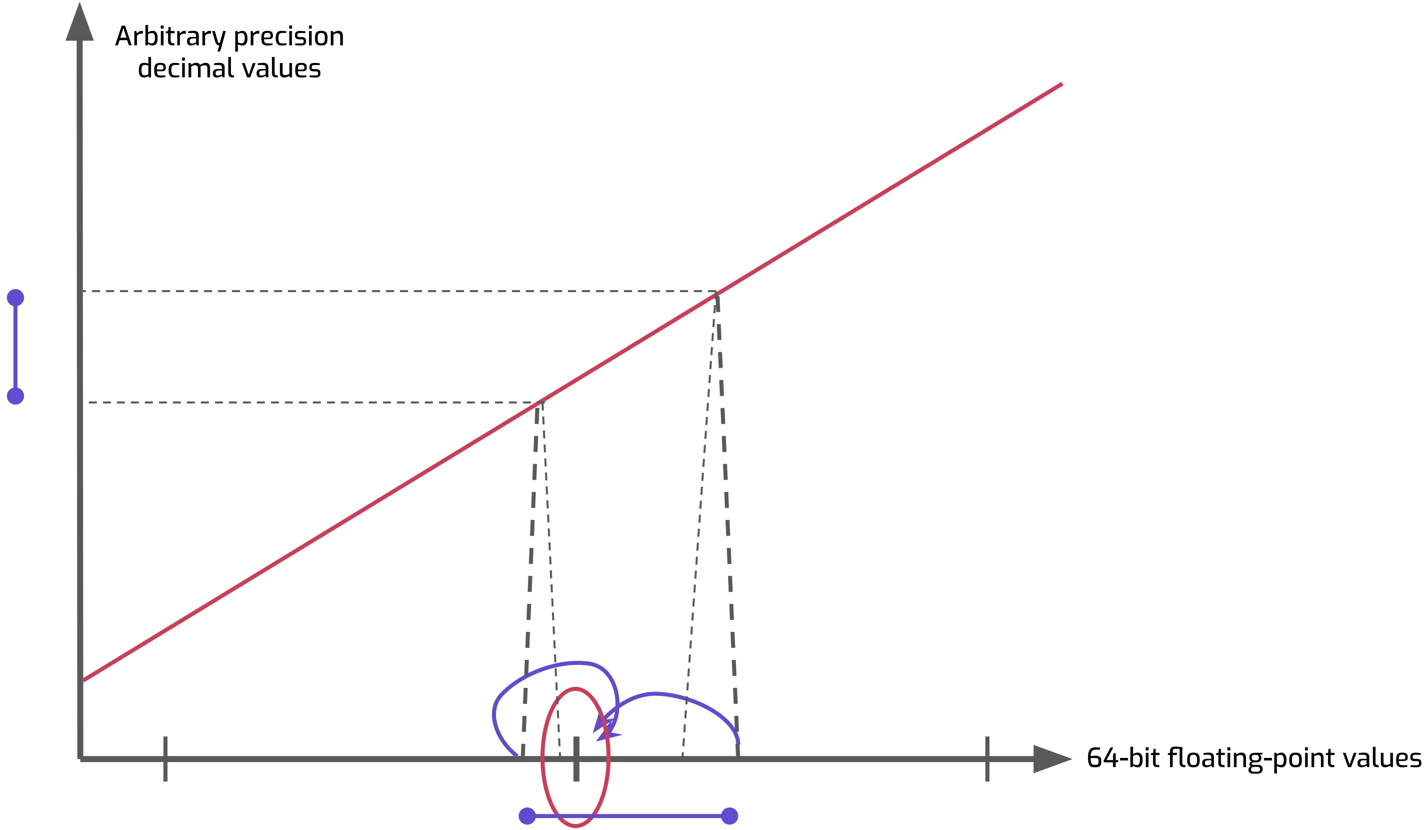
Four core ideas



Idea 4: use **arbitrary precision** values with **interval arithmetic**

Arbitrary precision floating-point values
64-bit floating-point values
Arbitrary precision floating-point values

Four core ideas


Arbitrary precision decimal values
64-bit floating-point values

Why this is neat

- **Simple security proof**: “just like” infinite-precision sampling + rounding! 💡
- **Fully generic**: works with many distributions, adapts to other methods! ✨
- **Fast**: converges quickly, especially if we generate many bits at a time 🏎️

Takeaways

- Differential privacy can have **vulnerabilities**! 😱
- To fix them, ad hoc approaches are **not robust enough** 🚫
- But **principled approaches** can be simple (and fast) enough! 🎉
- What do you need to do? **Nothing** – just use a library with a proven fix 😇

Shout-outs

- Authors of diffprivlib, SmartNoise Core & OpenDP for quickly acknowledging the vulnerabilities ❤️
- Authors of OpenDP for fixing the vulnerabilities 💙
- Authors of Google's DP library, for implementing another approach that comes with a privacy proof and isn't vulnerable to these attacks 💛
- Everyone who ships open-source code allowing this kind of research 💚

Thank you 💖

Stay in touch!

Damien Desfontaines
damien@desfontain.es
@TedTed@hachyderm.io

Learn more!

About us: tmlt.io
About our code: tmlt.dev