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

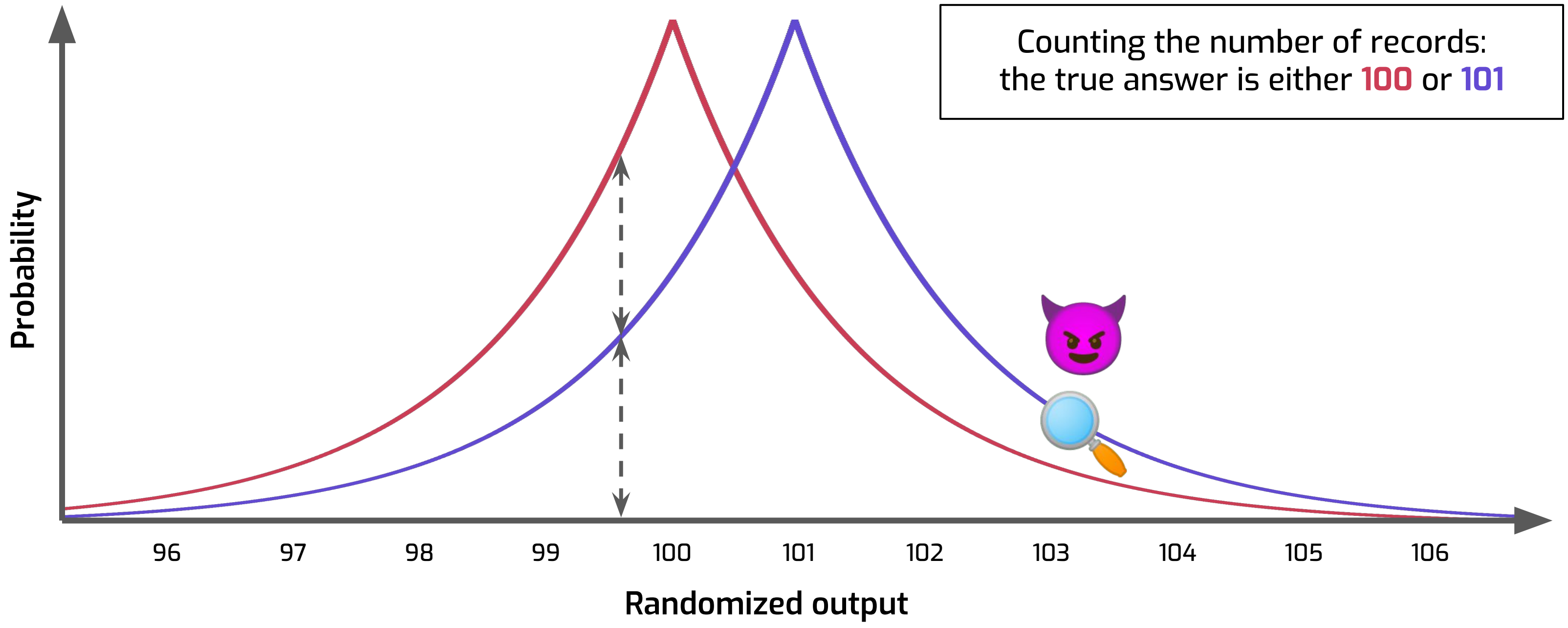
Or: what do you do when x + y = privacy vulnerability?

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

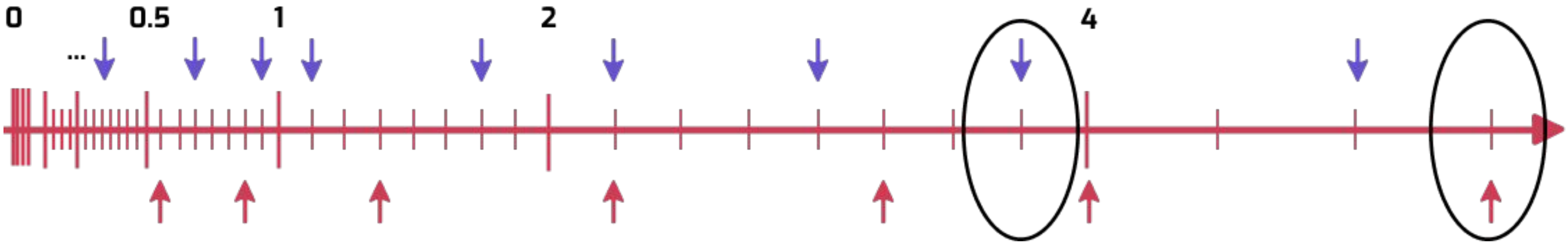
Samuel Haney
sam.haney@tmlt.io

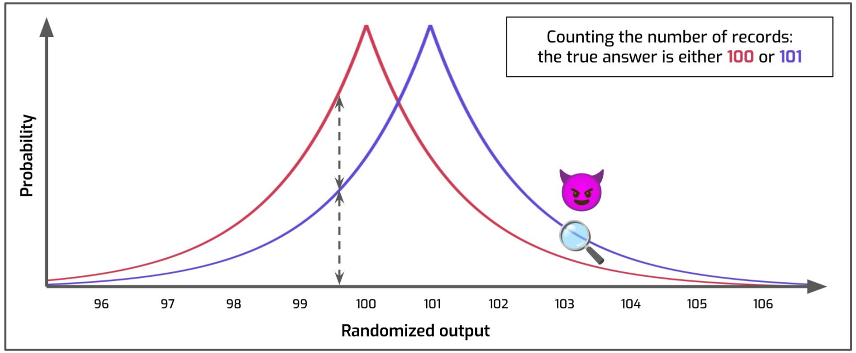
Differential privacy in one slide

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



What happens to our continuous line?





Why does this happen?

```
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!

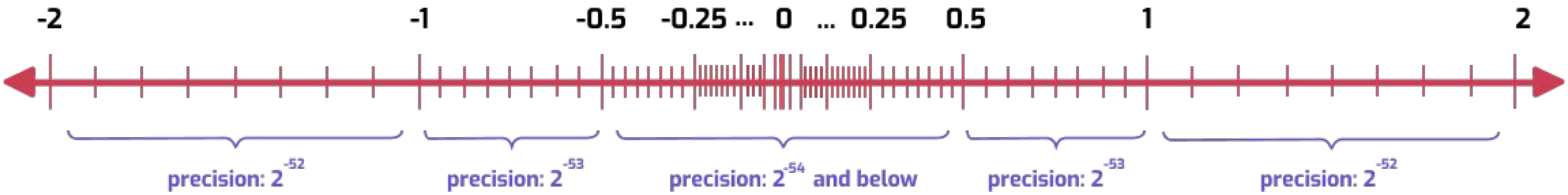
```
def add_noise(true_value, epsilon):
  sign = random.choice([-1, 1])
  u = random.uniform(0, 1)
  noise = sign * math.log(u) / epsilon
  return true_value + noise 😈
```

But… what about the
sum at the very end?

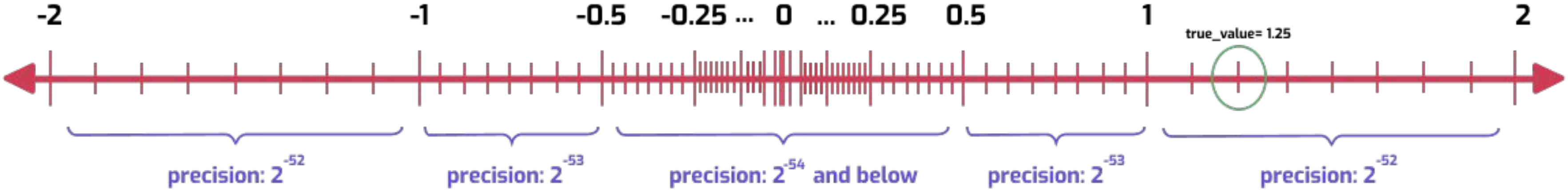
Attempt 1: fixing the noise generation
to get a distribution without "holes".

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

Fun fact about floating-point addition...

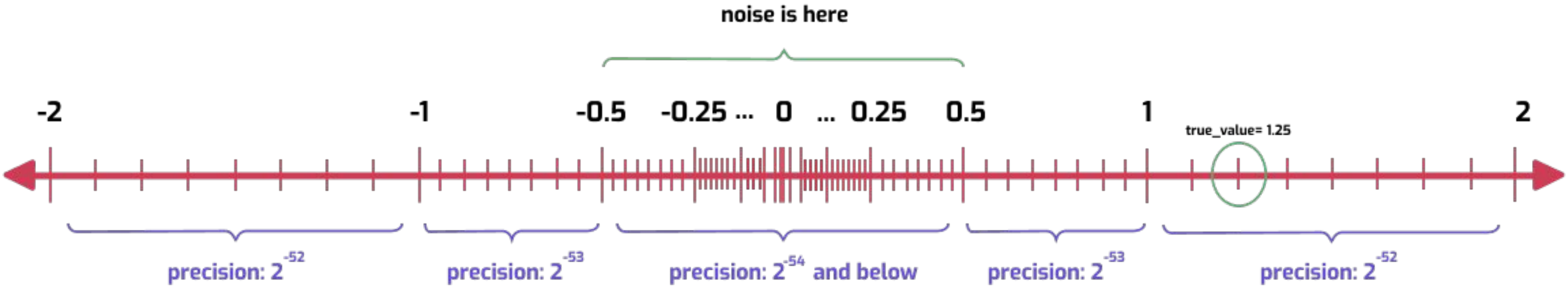


Fun fact about floating-point addition...



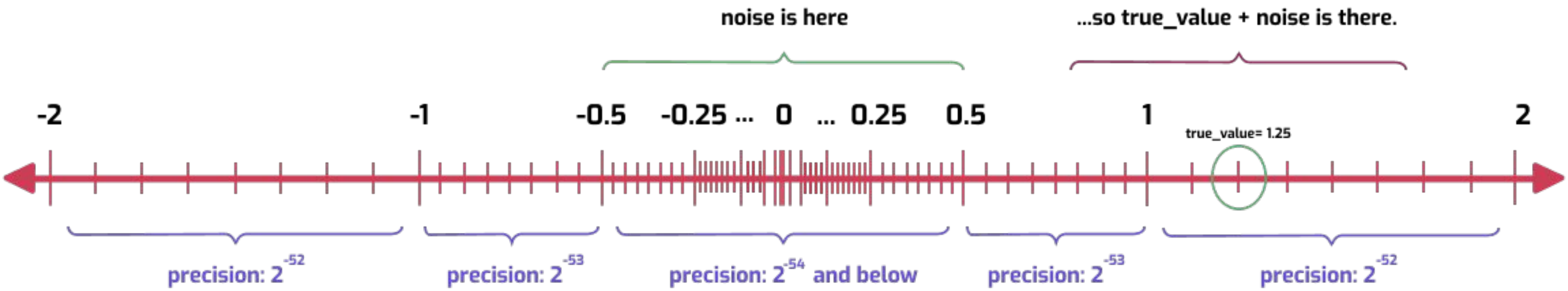
What if we add noise to 1.25?
It has precision 2^{-52}.

Fun fact about floating-point addition...



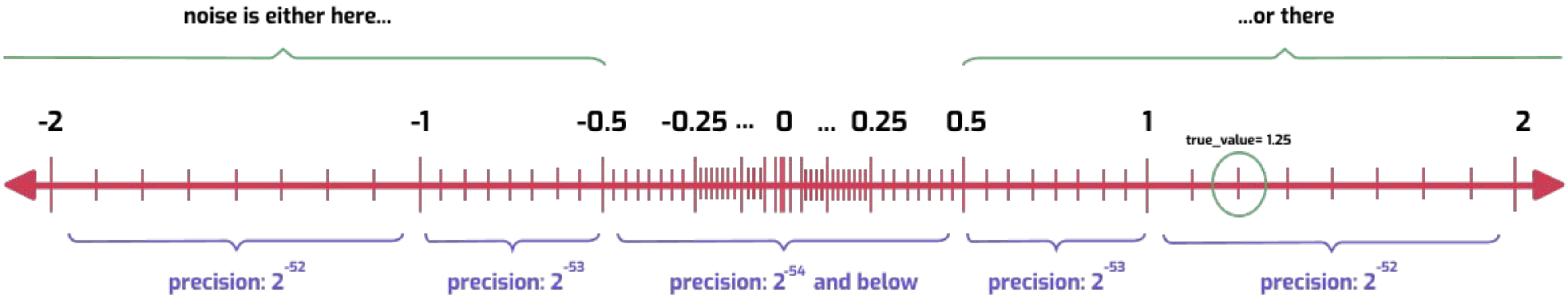
If the noise is small...

Fun fact about floating-point addition…



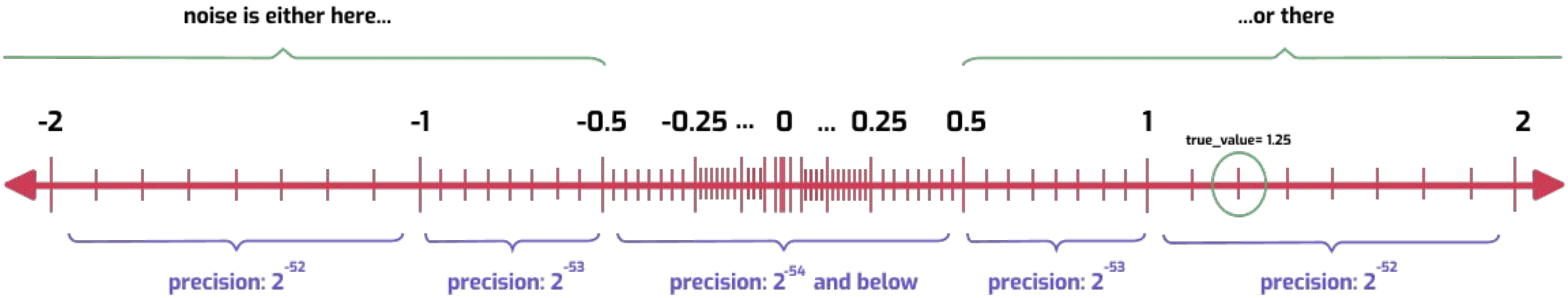
If the noise is **small**…
the sum's precision is at least 2^{-53}.

Fun fact about floating-point addition…



If the noise is **large**…

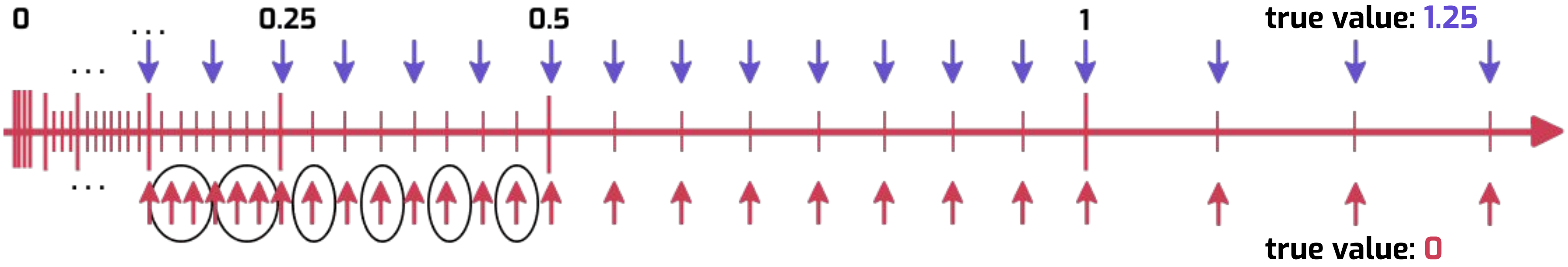
Fun fact about floating-point addition…



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}**.

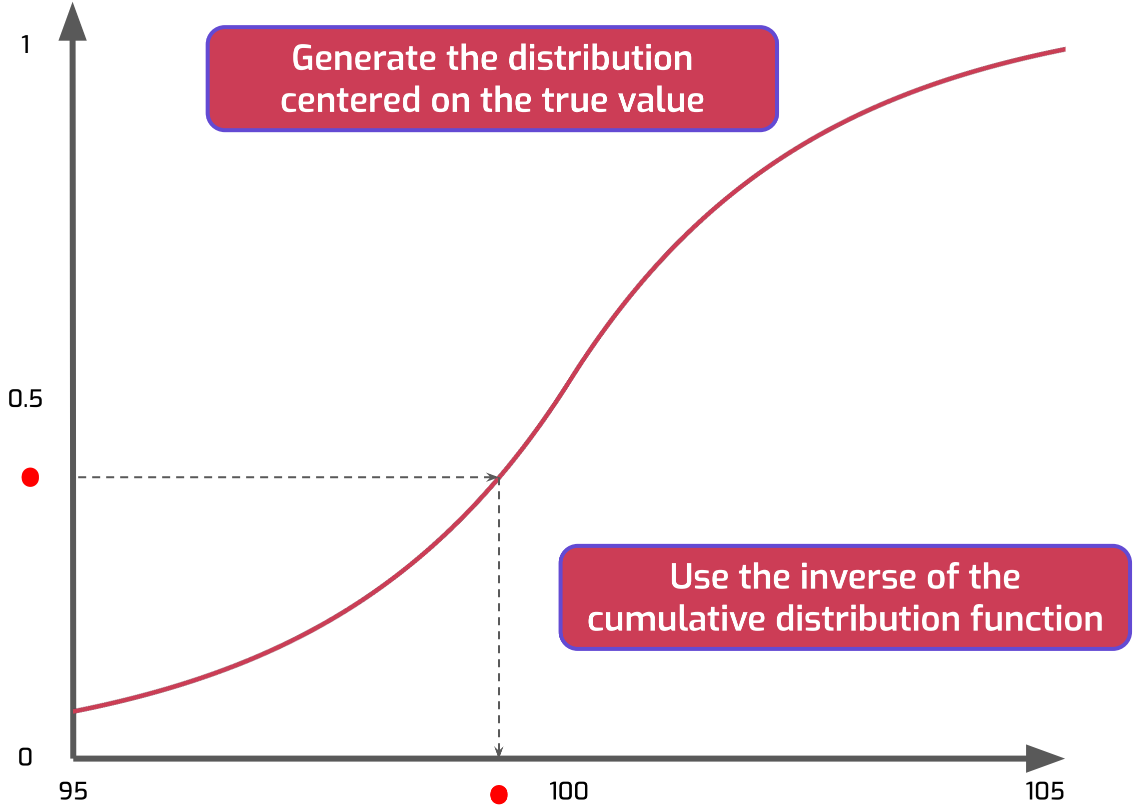


How do we fix it?

```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
```

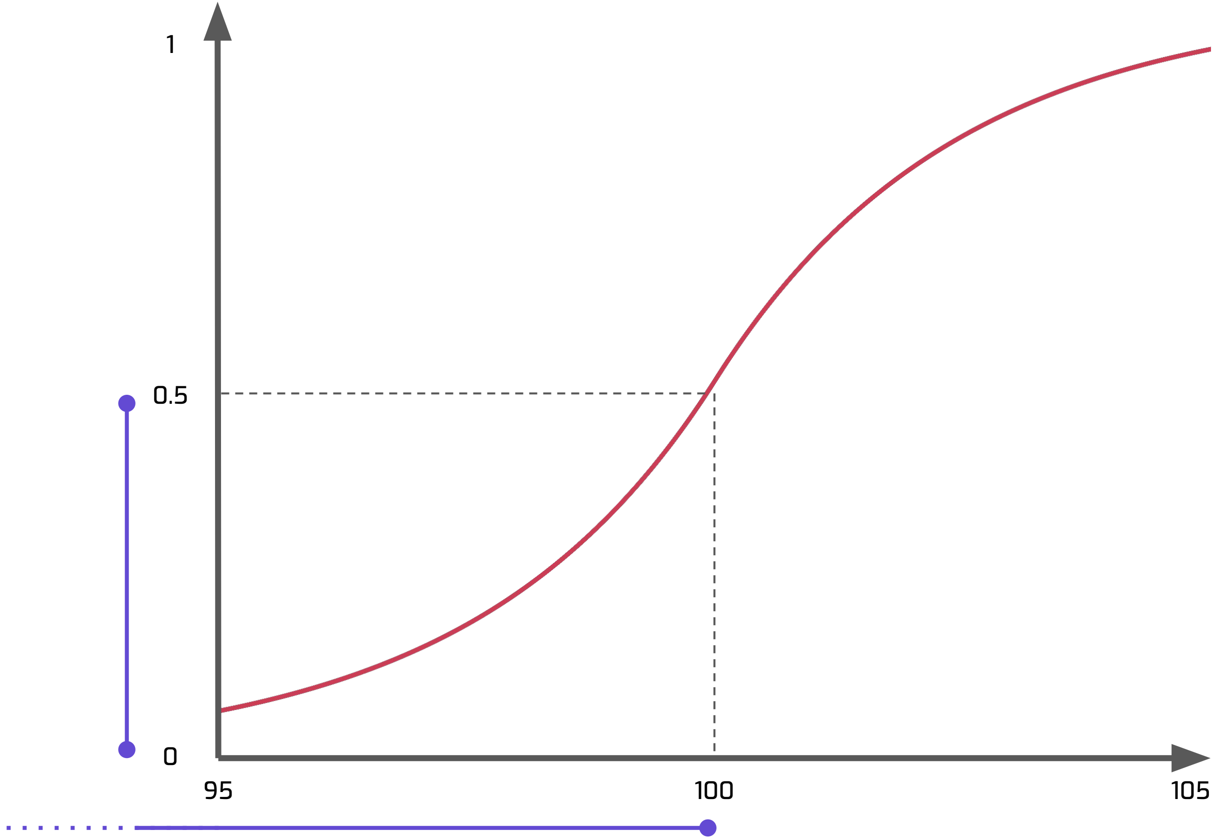
We need to fix the entire routine, not just the noise generation!

General aim

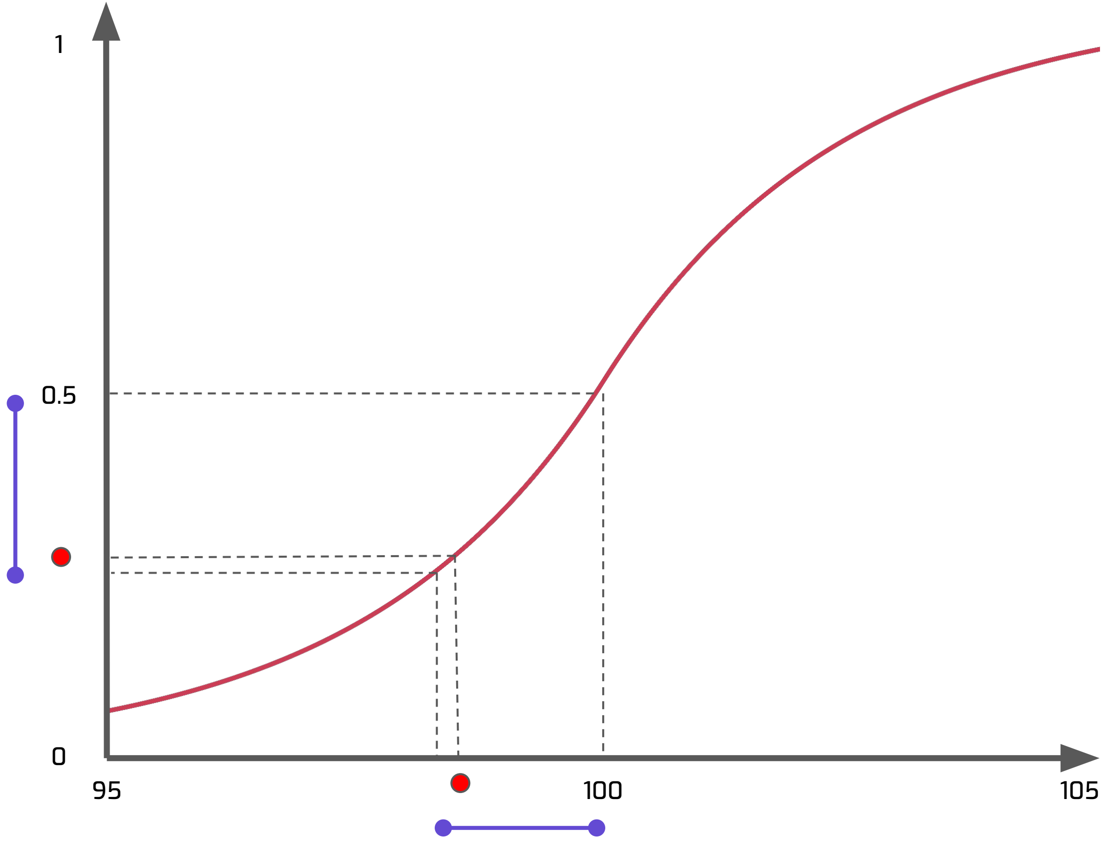


1
0.5
0
95
100
105
Generate the distribution centered on the true value
Use the inverse of the cumulative distribution function

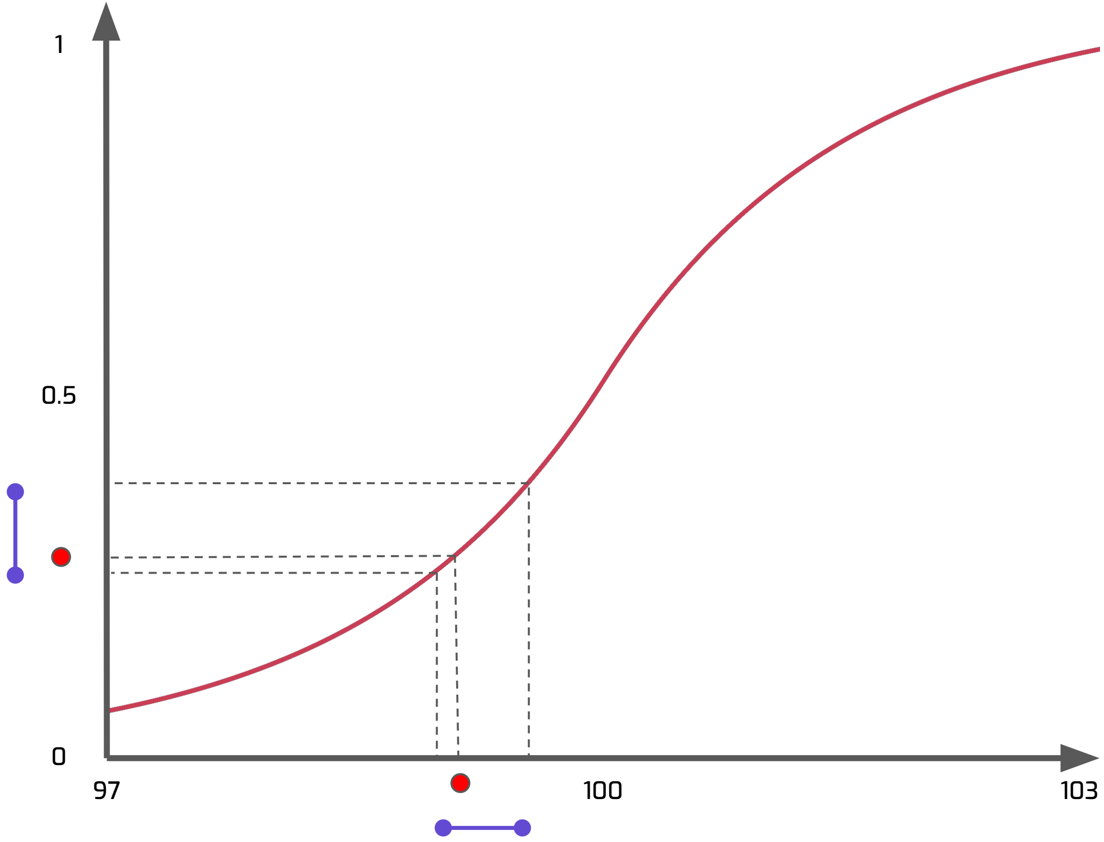
Sample intervals instead



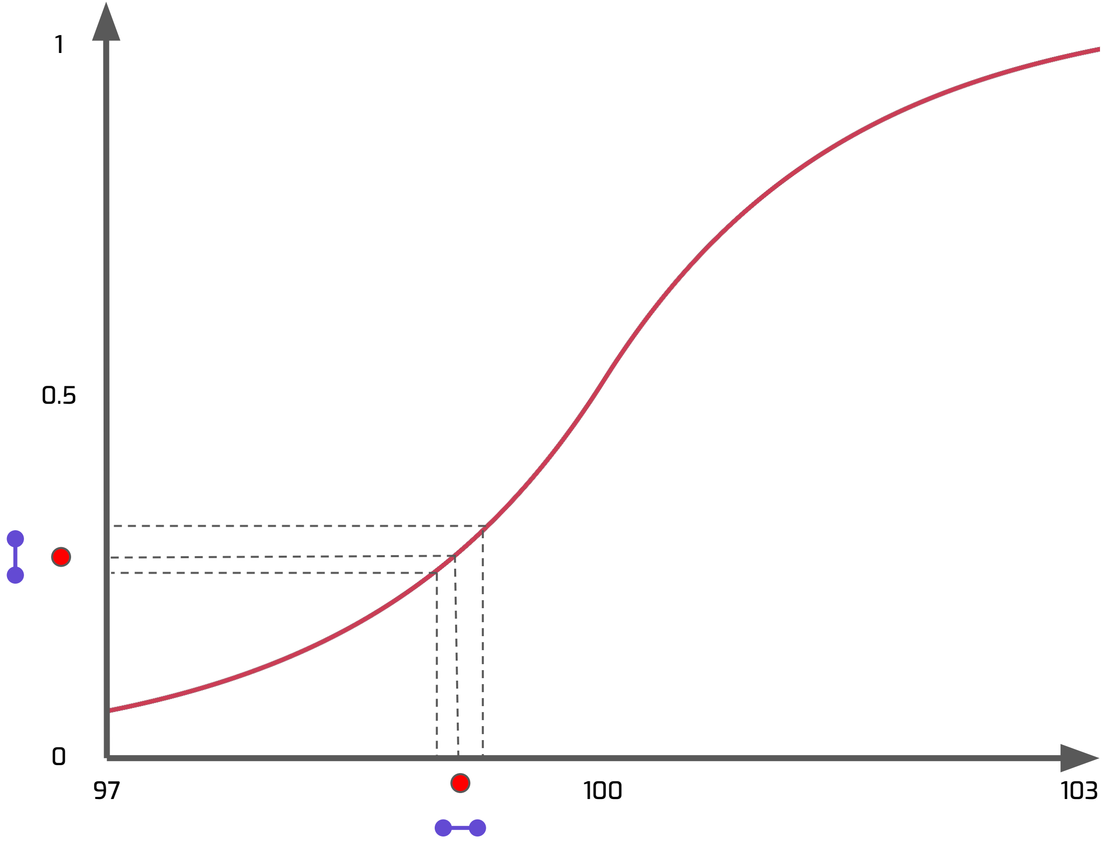
Interval refining



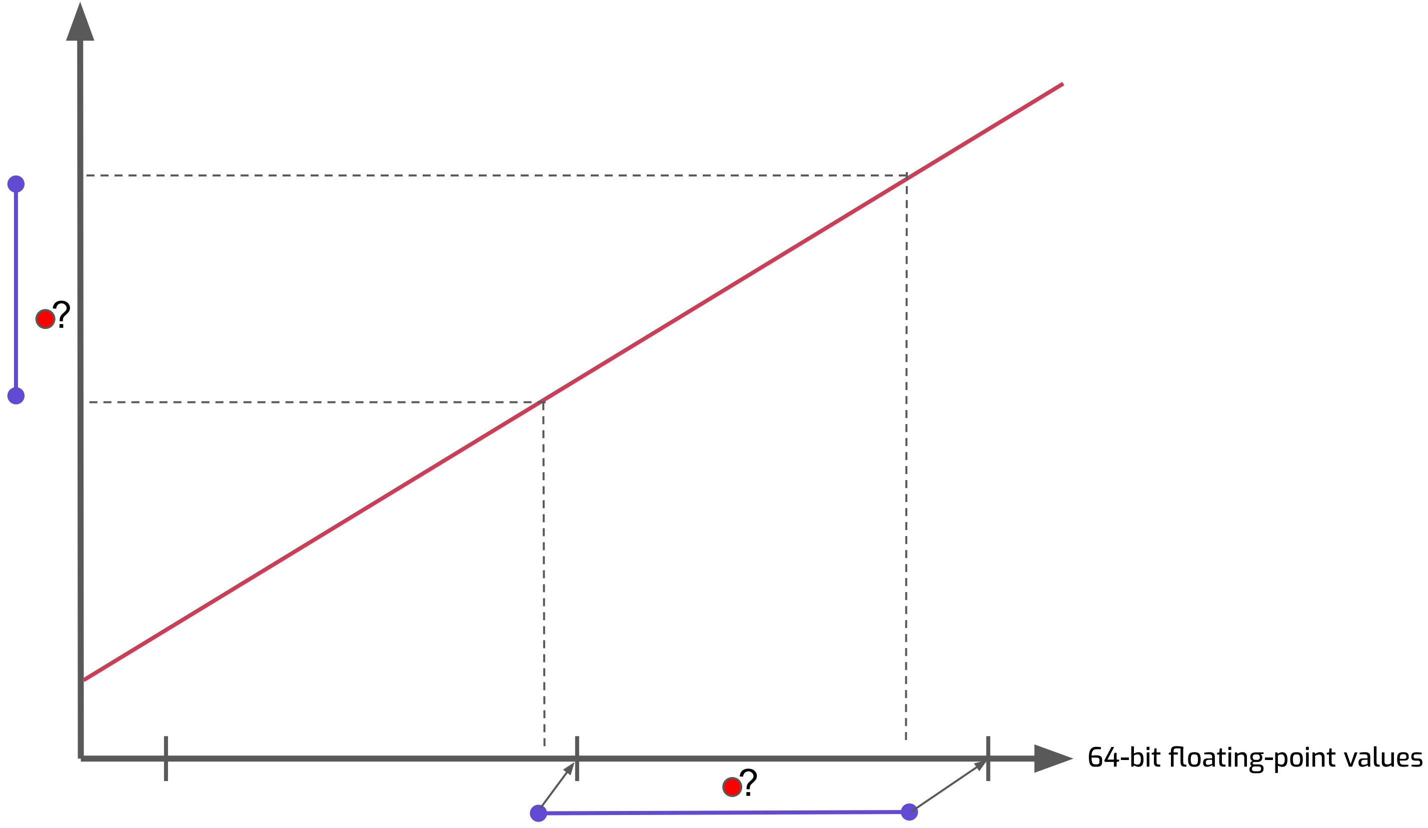
Interval refining



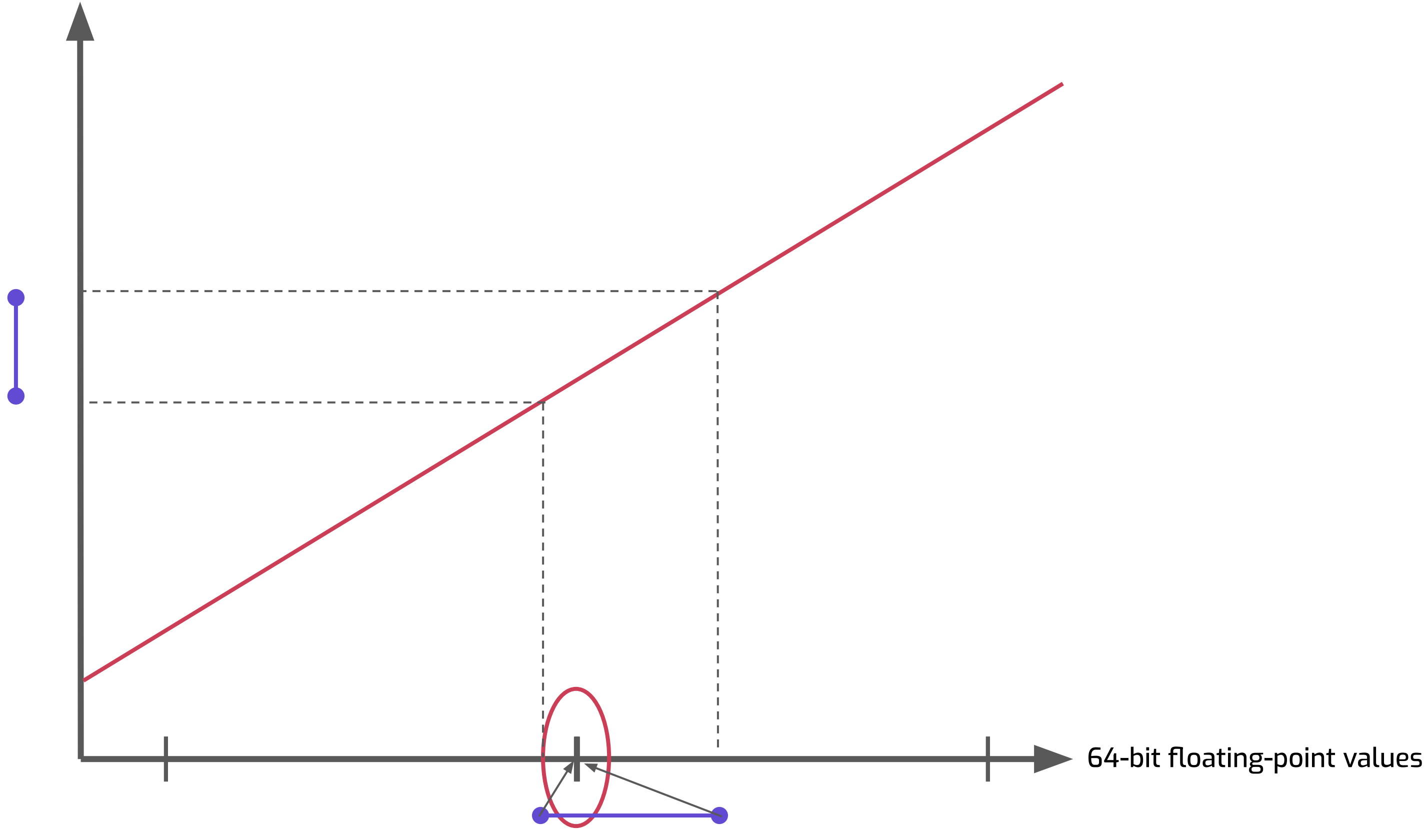
Interval refining



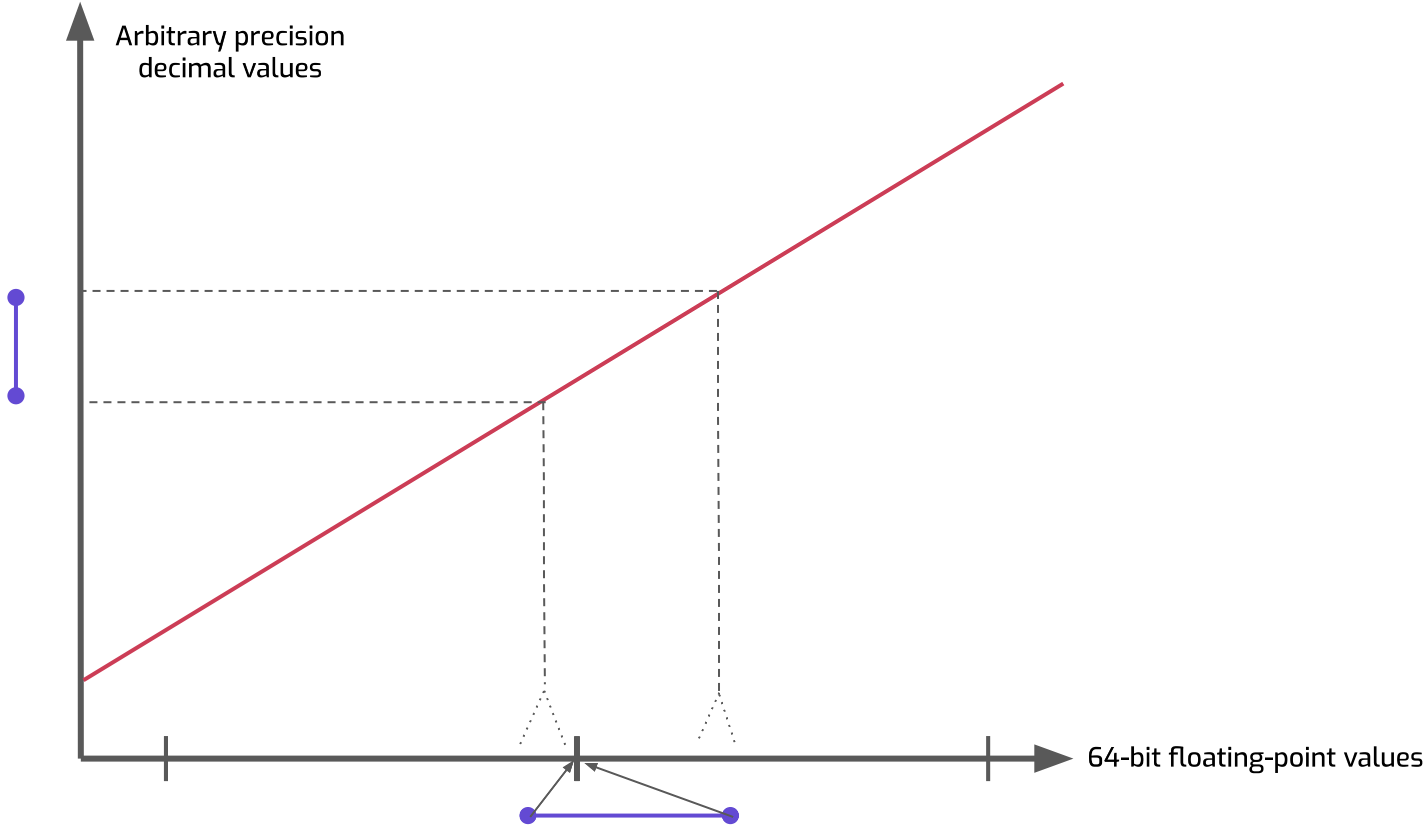
Rounding the interval



Termination condition



One more detail... interval arithmetic



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 😇

Impact & mitigations

Library	Status	Comments
SmartNoise Core	Vulnerable, won't fix	Project was deprecated
Diffprivlib	Vulnerable, not fixed	Snapping mechanism available for Laplace, no fixes for other distributions
OpenDP	Vulnerable, then fixed	Configurable discretization parameter, doesn't generalize
GoogleDP	Not vulnerable	Fixed discretization parameter, small privacy cost, doesn't generalize
Tumult Analytics	Not vulnerable	Hyperparameter-free, generalizes ✨

Thanks to everyone who ships open-source code 💚

Thank you 💖

Stay in touch!

We're Sam Haney and Damien Desfontaines on the PEPR Slack 🌶️

Learn more!

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