Privacy-Preserving Systems (a.k.a., Private Systems)

CU Graduate Seminar

Instructor: Roxana Geambasu

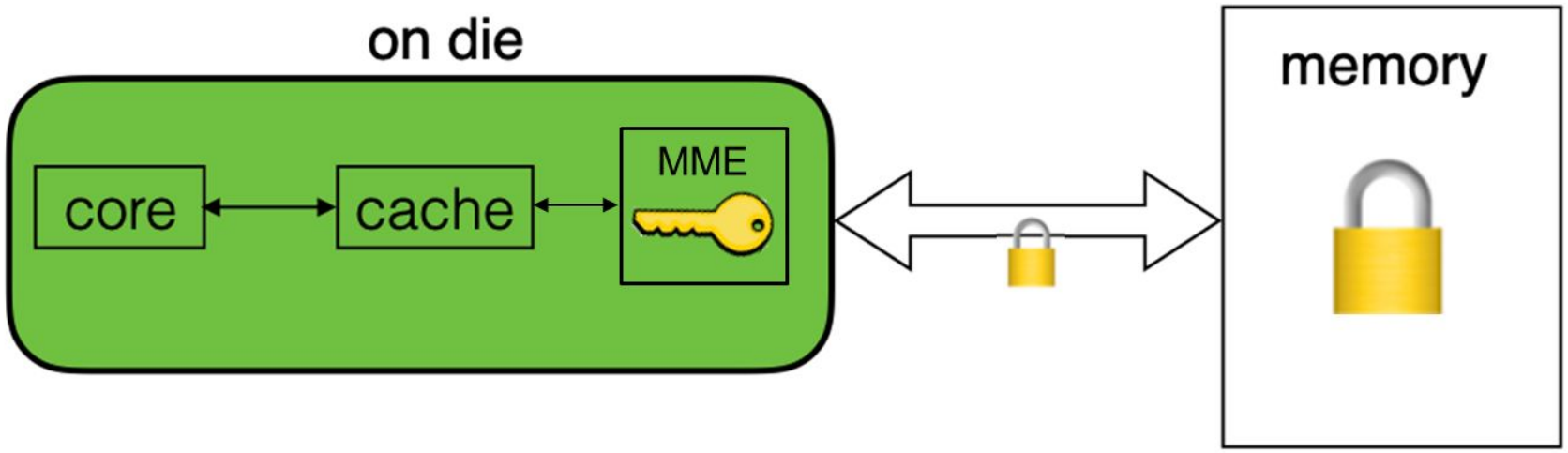
Other Privacy Technologies

- Hardware enclaves
- Secure multi-party computation
- Federated learning

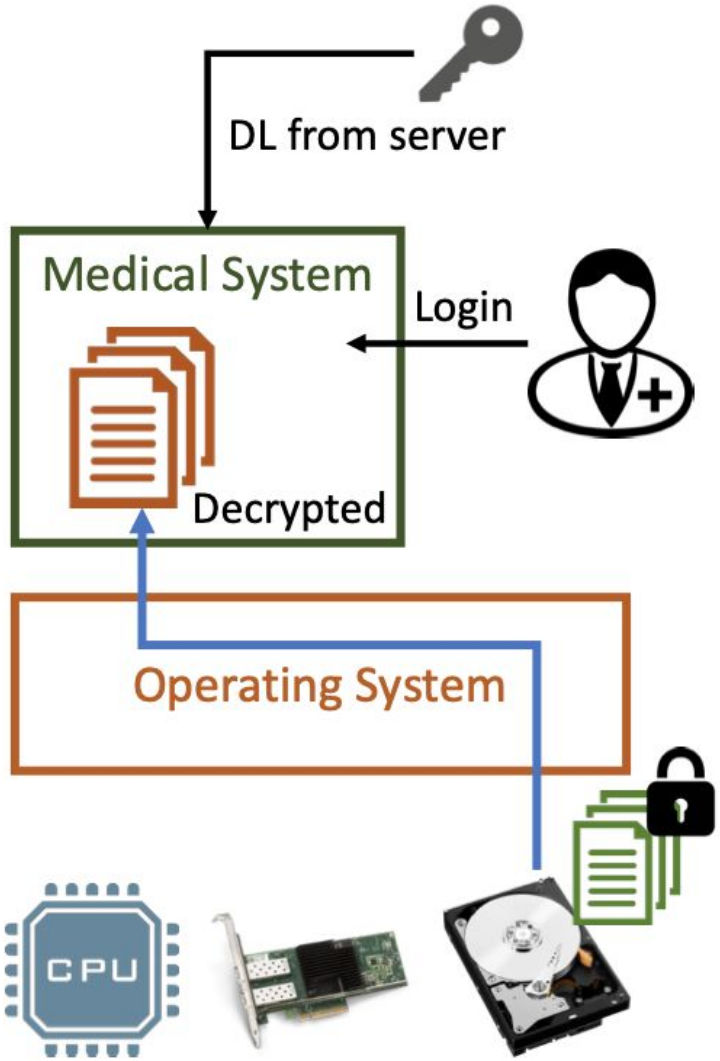
Hardware Enclaves

Hardware Enclaves (e.g., Intel SGX)

- Hardware-enforced isolated execution environment
- Data decrypted only on the processor
- Protect against an attacker who has root access or compromised OS
- Cloud offerings: Azure Confidential Computing, Google Asylo, …

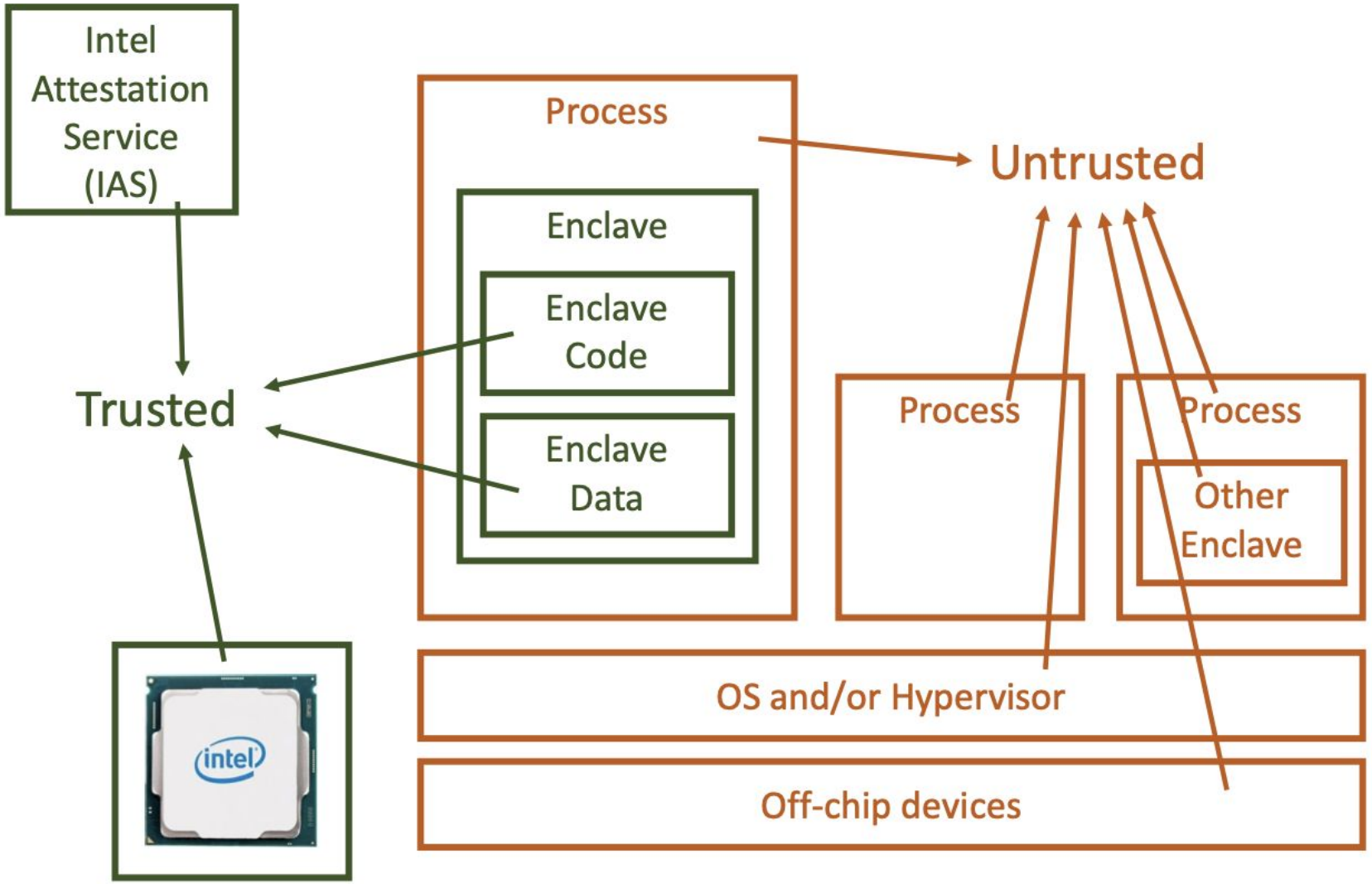


System Threats to Trusted Execution



- What can go wrong?
  - Side channels
    - out of scope for Intel SGX
  - Counterfeit software
  - Inject rootkits into OS
  - Privilege escalation
  - Install malicious kernel
  - Compromised HW devices
  - Cold-boot attacks

Threat Model for Hardware Enclaves

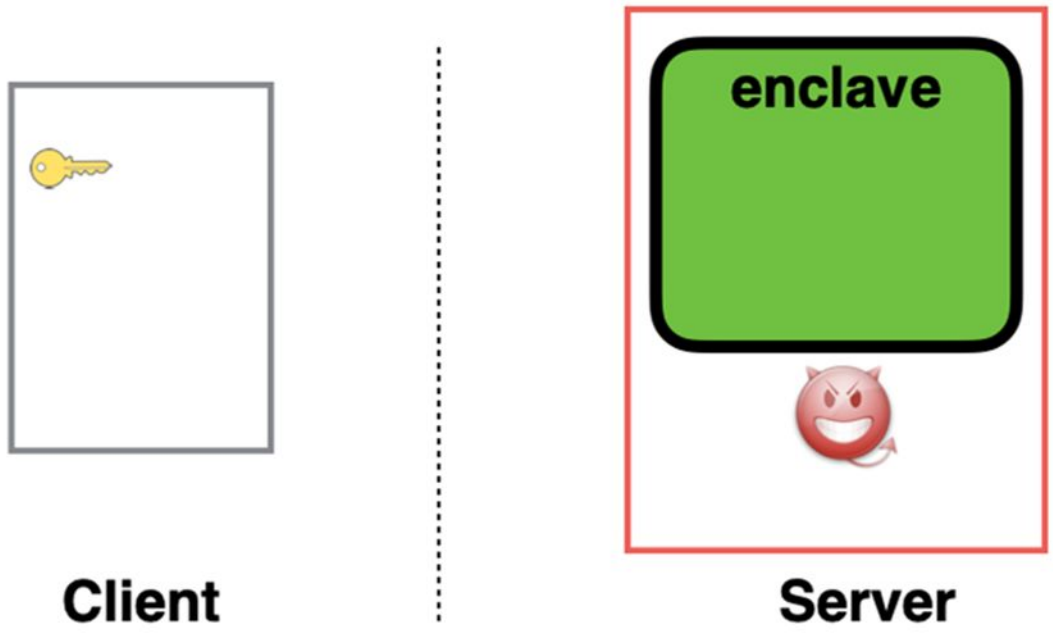


Elements of Secure Enclaves

- Secure boot: HW-verified measurement + first instruction
- On-chip program isolation
- Cryptographically protected external memory
- Execution integrity; no interference from attackers
- Remote Attestation
- Secret sealing

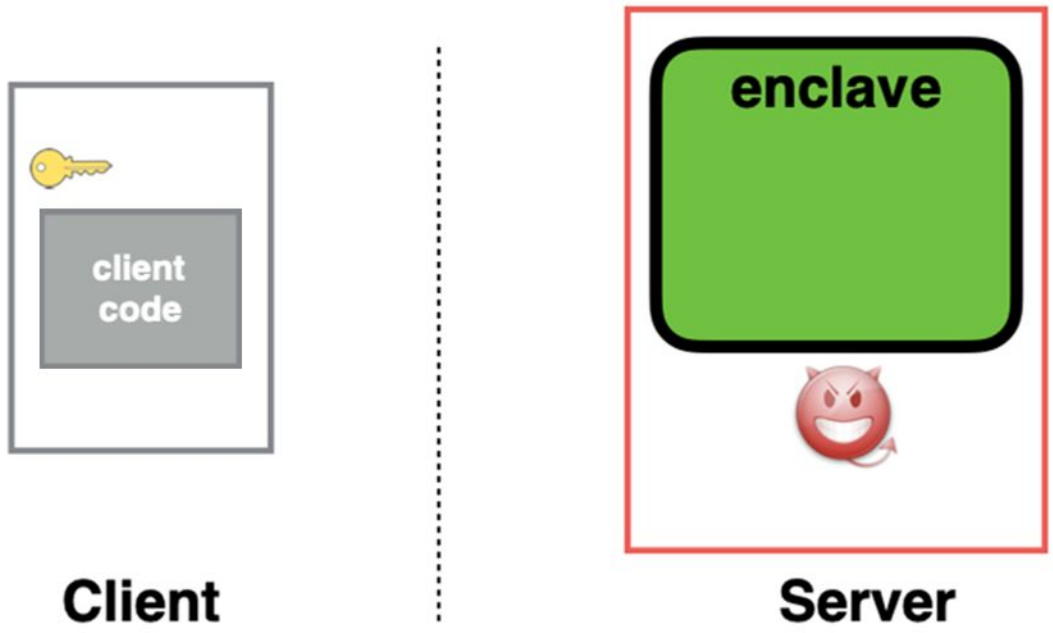
Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.



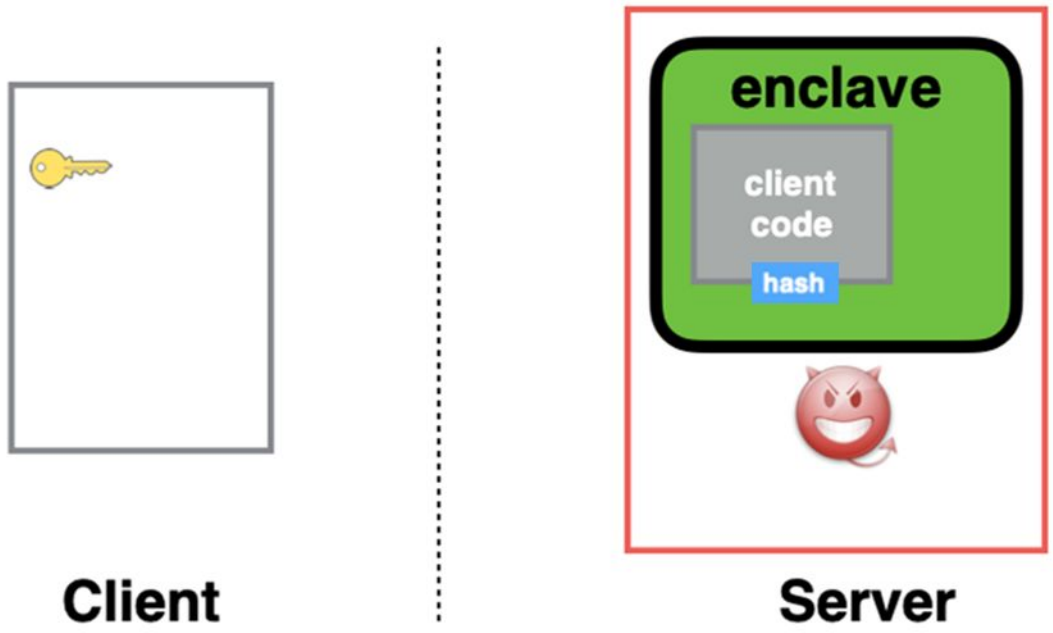
Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.



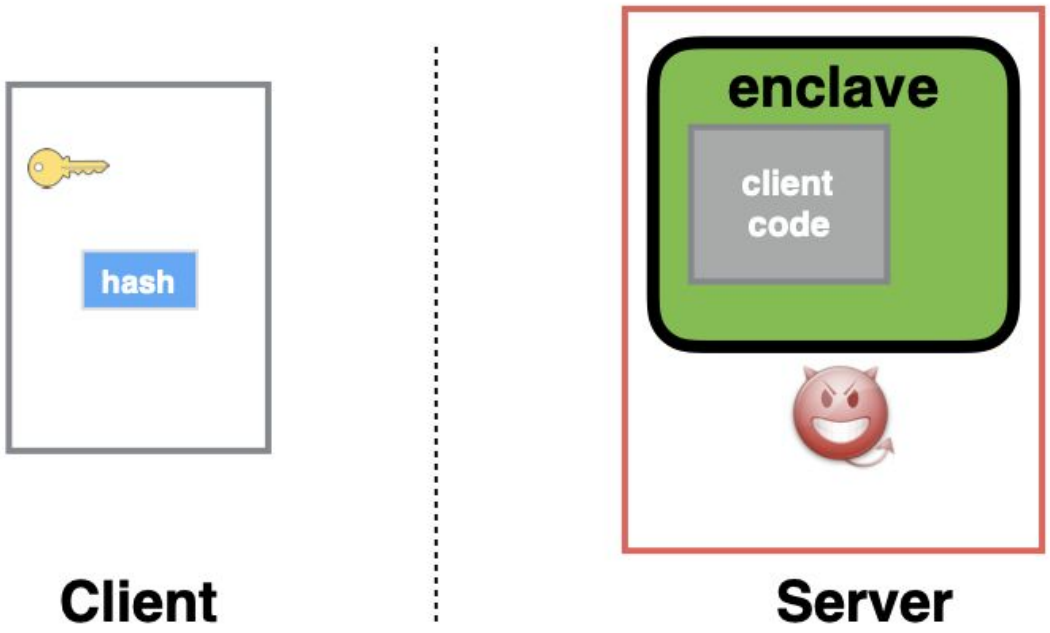
Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.



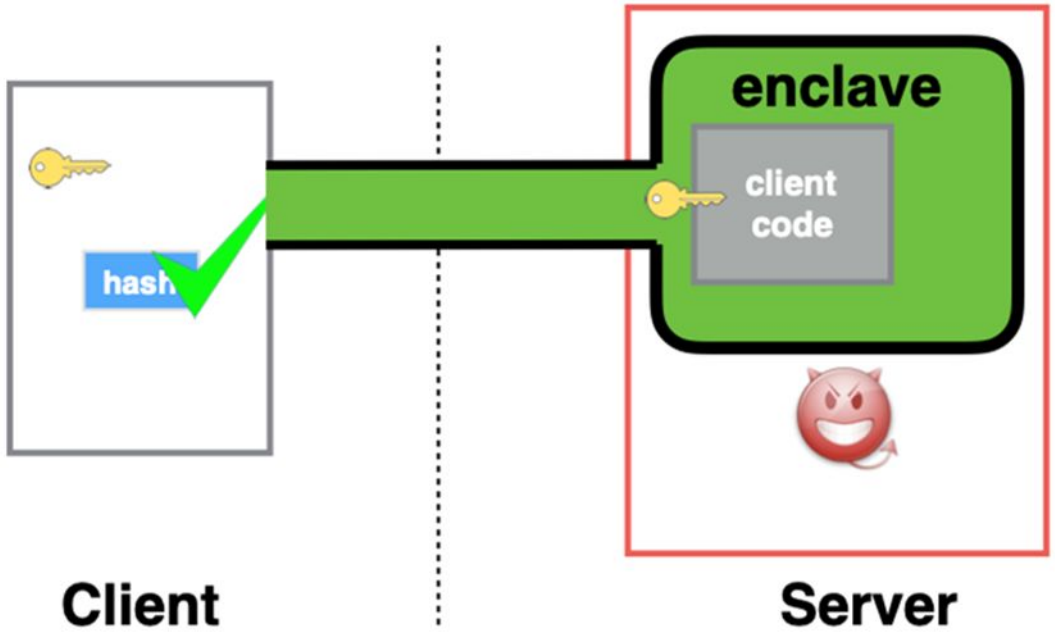
Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.



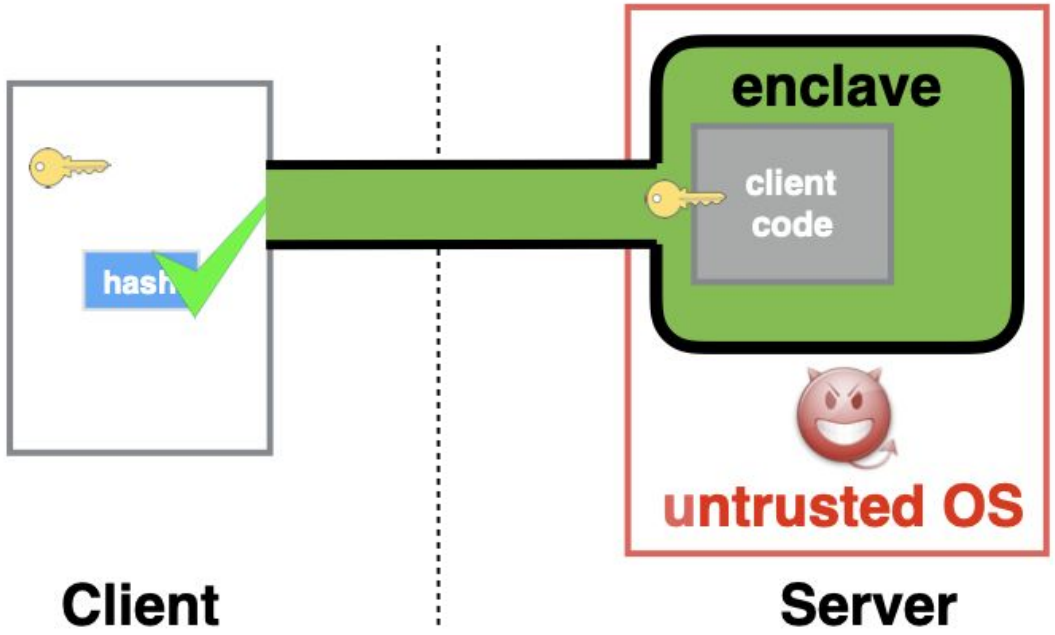
Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.

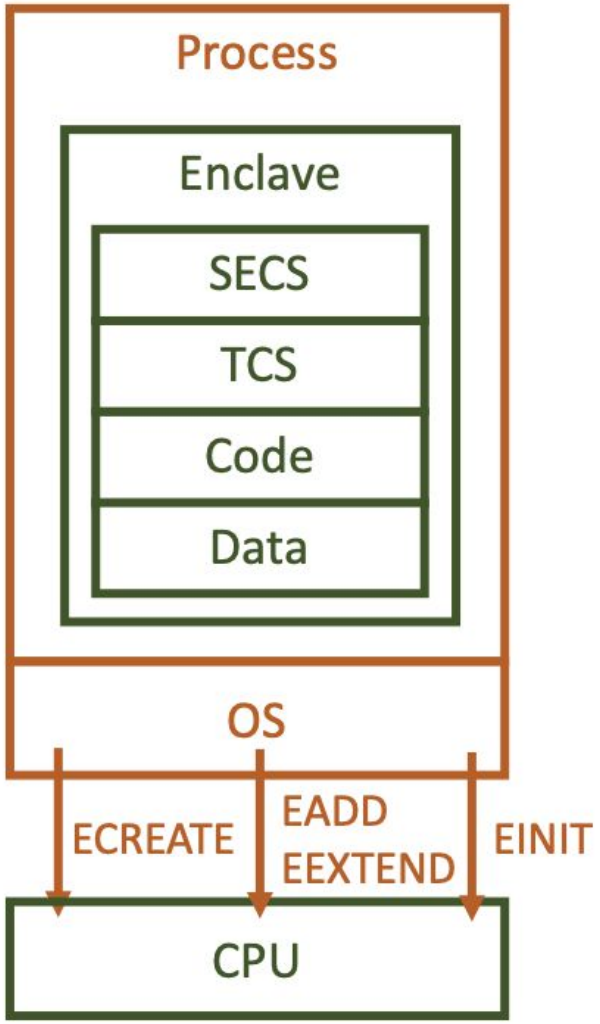


Remote Attestation

Enables verifying which code runs in the enclave and performing key exchange. With this, you can bootstrap end-to-end encryption between your clients and the authenticated (trusted) code of your application.

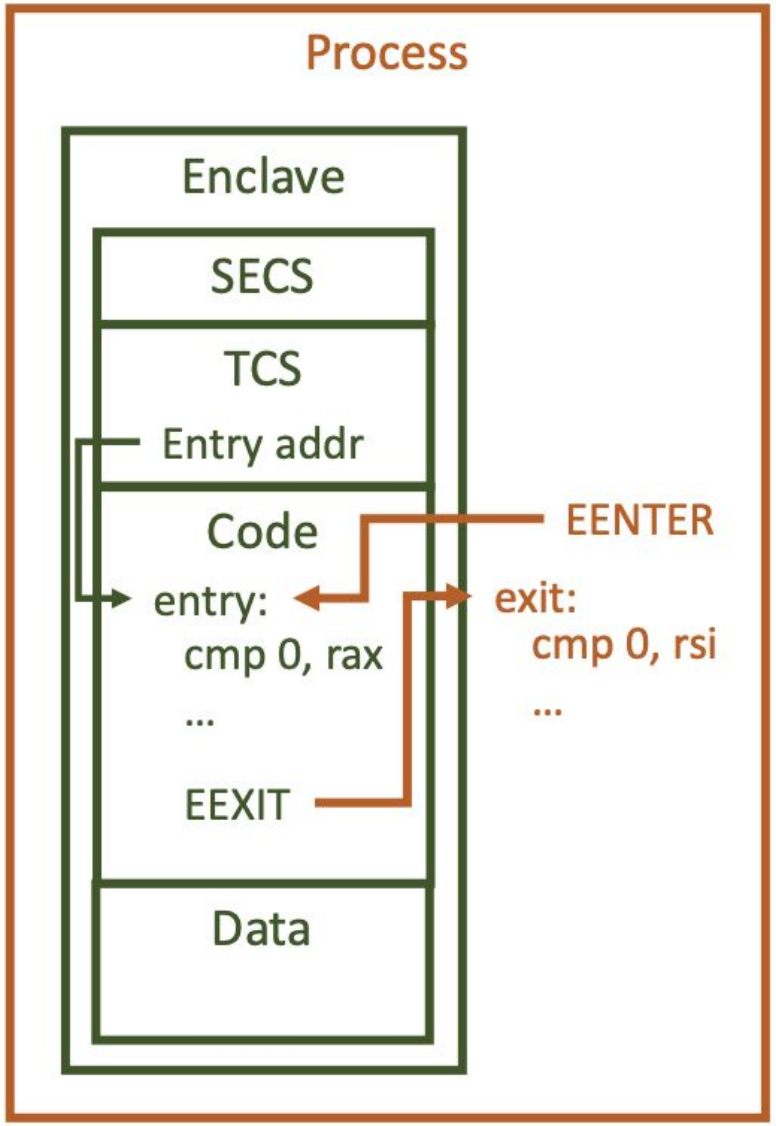


Enclave Creation with Intel SGX



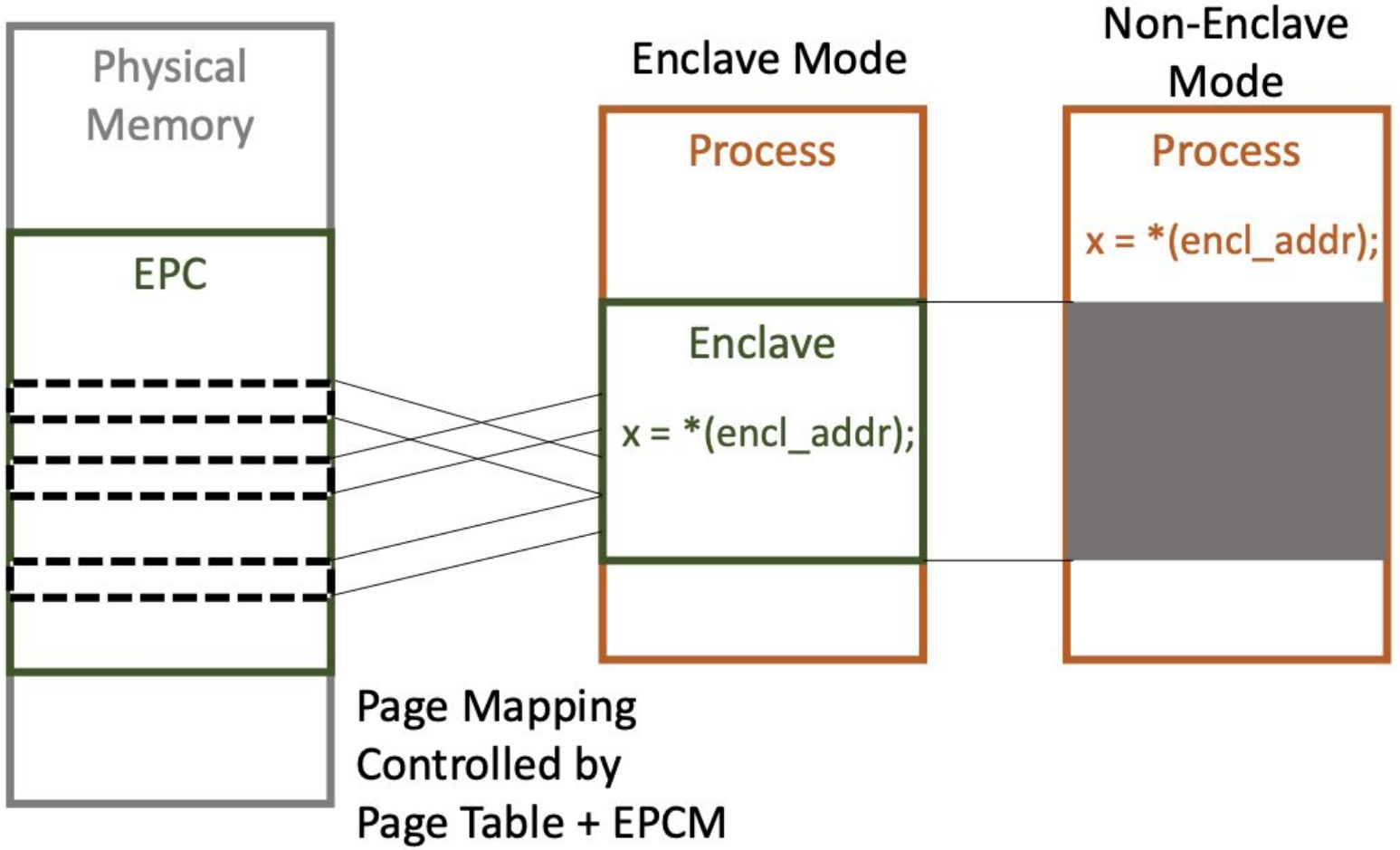
- ECREATE(SECS):
  create an enclave range
- EADD(SECS, addr, prot),
  EEXTEND(SECS, addr):
  add a page to enclave and measure the content
- EINIT(SECS, license):
  check & initialize an enclave

Enclave Enter & Exit



- EENTER(SECS, TCS):
  enter at a static enclave addr
- EEXIT(addr):
  exit enclave to any addr
- Enclave can accept parameters after the entry
- Attackers cannot interfere control flow unpredictably

Enclave Isolation



Abort page semantic:

EPC pages contains all 0s for execution outside the enclave

Existing Systems

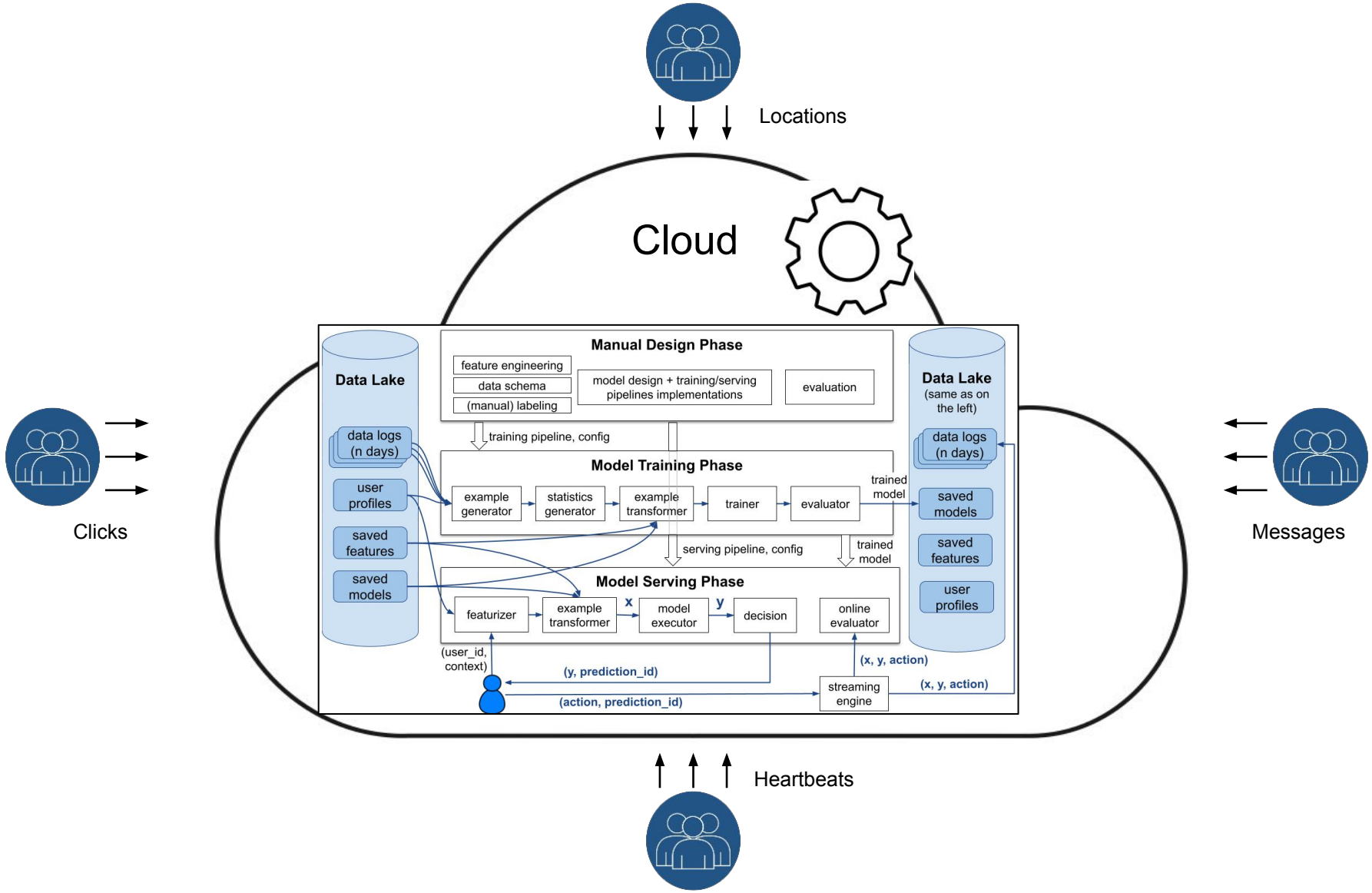
Hardware enclaves are a very real technology that is available in multiple clouds, e.g.:

- Amazon: AWS EC2 Nitro Enclaves
- Microsoft: Azure SGX Enclaves
- Google: GCP Asylo

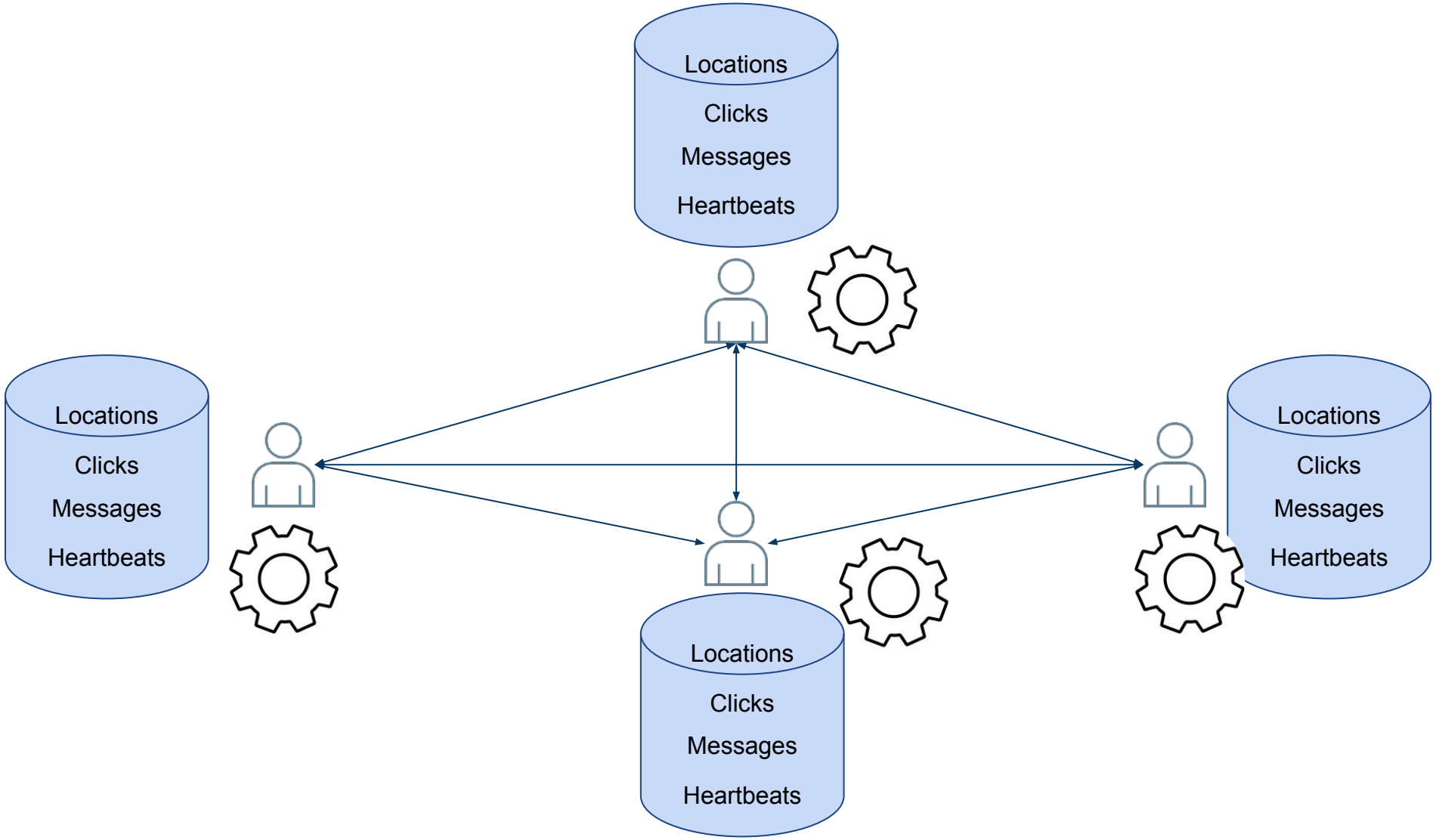
The End

Private Collaborative Learning

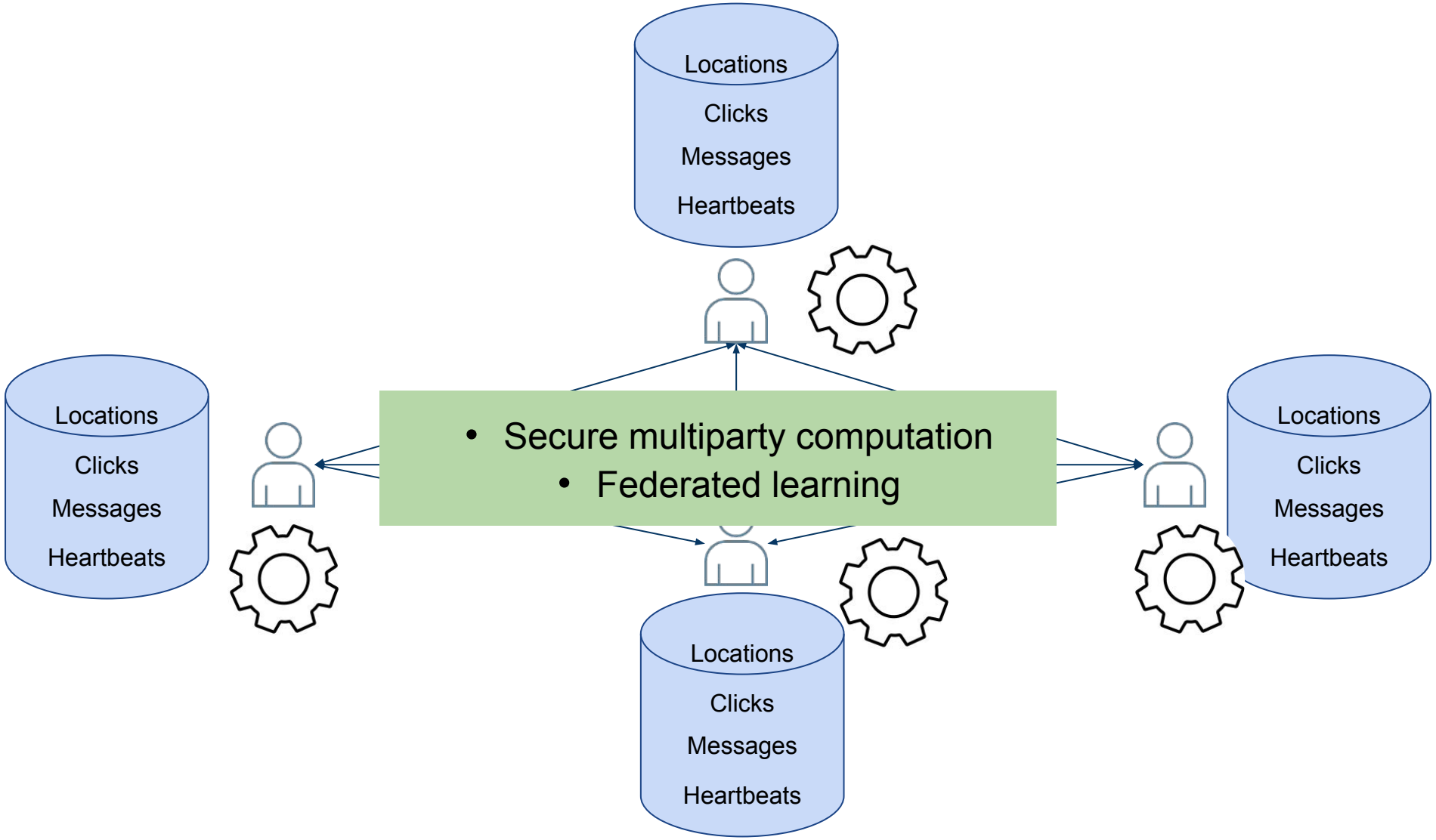
What If No Central Aggregation of Data?



What If No Central Aggregation of Data? (cont.)



What If No Central Aggregation of Data? (cont.)



Case 1: Money Laundering Detection

- Banks want to detect money laundering using machine learning.
- Criminals conceal illegal activities across many banks.
- Banks want to jointly compute a model on customer transaction data, but cannot share data.

Secure Multiparty Computation

- Parties emulate a trusted third party via cryptography.
- No party learns any party's input beyond the final result (trained model).
- Performance is a challenge, but for simple computations (such as computing linear models) and few parties (up to 10), this is practical.

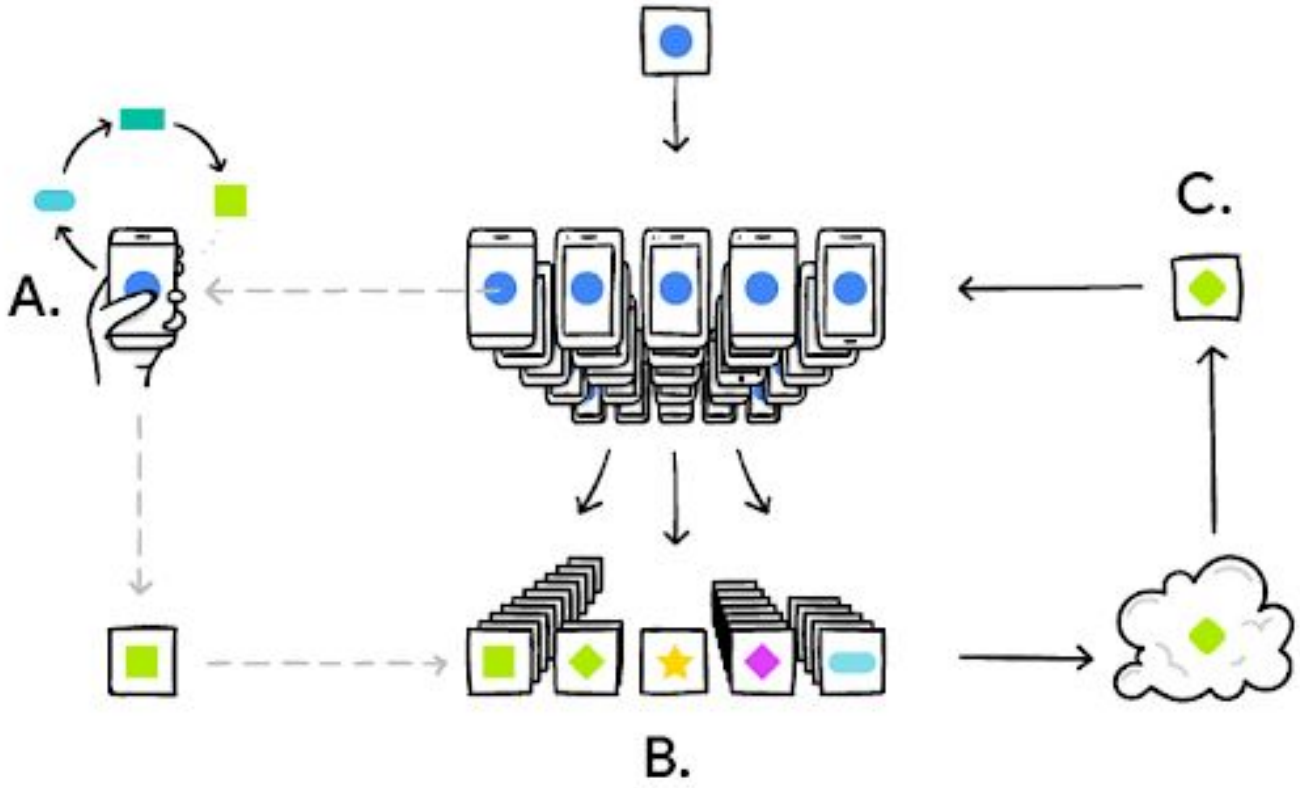
[Yao82]

Case 2: Text Autocomplete

- Want to train a text autocomplete model on many users' data but don't want to collect users' data in a central location.
- Each user trains a local, partial model, and then the cloud combines these models into a global model, which it ships back to the clients.

Federated Learning

- Your phone personalizes the model locally, based on your usage (A)
- Many users' updates are aggregated (B) to form a consensus change (C) to the shared model
- The procedure is repeated as new data becomes available



Credit: Google AI blog

Existing Systems

MPC and FL are both practical. Here are a couple (of multiple!) example offerings:

- Inpher's XOR Secret Computing
- Google's Tensorflow Federated

Cited References

(Yao82) Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, 1982.

The End

MPC Details and Demo

by Pierre Tholoniat

Introduction

- General MPC setting
  - Multiple parties with private inputs
  - Emulate a trusted party to compute a function on their inputs
  - Without revealing anything else than the output
- How do MPC protocols work? How practical are they?
  - Pretty informal presentation
  - See the Pragmatic MPC textbook [1] and other references for details and proofs

Introduction

Two main threat models:

- Honest-but-curious adversary
  - Corrupt parties follow the protocol, but try to learn as much as they can
  - A.k.a passive or semi-honest adversary
- Malicious adversary
  - Corrupt parties can deviate from the protocol arbitrarily
  - A.k.a active adversary

- Today, we consider an honest-but-curious adversary
  - Simple setting to show essential techniques
  - Protocols can be converted from passive to active security

Outline

1. Shamir Secret Sharing
2. Evaluating Arithmetic Circuits with the BGW Protocol
3. MPC with Preprocessing: Beaver Triples
4. Implementation: Meta’s Private Computation Framework

1. Shamir Secret Sharing

Shamir, 1979 [8]

Setting:

- n parties, threshold $t \leq n$
- A global secret $y \in K := F_p$ is shared among parties
- Each party i has a share y_i
- Notation for a sharing of y: $[y] := (y_1, \ldots, y_n)$

Desired properties:

- Knowing $k \geq t$ shares is sufficient to reconstruct y
- Knowing $k < t$ shares doesn't reveal anything about y

How can secret-sharing be useful?

Example: secret key recovery

- Split your wallet key into n=5 backups servers
- Reconstruct the key from t servers when needed
- If t=1, a single corrupted server can steal your key
- If t=5, a single faulty backup prevents you from recovering your key
- If t=3, resilient against 2 corrupted colluding servers and 2 failures

We can also use secret-sharing for arbitrary MPC

Construction with polynomials

Lagrange interpolation:

- Fact: the only polynomial of degree ≤ *t-1* with *t* roots or more is zero
- Consequence: any polynomial $P \in K_{t-1}[X]$ is uniquely characterized by the list of coordinate pairs $(P(x_1), \ldots, P(x_t))$ for $(x_1, \ldots, x_t)$ distinct field elements
- Lagrange coefficients:

$$P(X) = \sum_{i=1}^{t} P(x_i) \prod_{j \neq i} \frac{X - x_i}{x_j - x_i}$$

Construction with polynomials

Protocol:

- We (the secret owner/dealer) sample a random polynomial in $K_{t-1}[X]$ such that $P(0) = y$
- Fix public non-zero interpolation points $x_1, \ldots, x_n$
- Distribute $y_i := P(x_i)$ to party $i \in \{1, \ldots, n\}$
- Any group of t parties can reconstruct y: $$y = P(0) = \sum_{i=1}^{t} P(x_i) \prod_{j \neq i} \frac{0 - x_i}{x_j - x_i} = \sum_{i=1}^{t} y_i \lambda_i$$

- The Lagrange coefficients λ_i can be computed in advance, we just need a linear combination of the shares to reconstruct the secret

2. Evaluating Arithmetic Circuits with the BGW Protocol

Ben-Or, Goldwasser and Widgerson, 1988 [9]

Can we perform operations on a secret-shared input?

- Example application: split a private key into *n* shares, and sign a document without ever reconstructing the private key locally
- Any computation in F_p can be represented as an arithmetic circuit (why?)
- We just need to have secret-shared version of the + and x gates

Using multiple inputs:

- In the Shamir setting we had a trusted dealer that splits a secret into shares
- The dealer can be a (semi-honest) party that shares its own input with other parties
- We run multiple Shamir sharings in parallel and combine them with gates

Addition Gate

- Two inputs shared with Shamir's scheme:
  - Secret p, polynomial P such that p = P(0), shares $P(x_1)$, …, $P(x_n)$
  - Secret q, polynomial Q such that q = Q(0), shares $Q(x_1)$, …, $Q(x_n)$
- Output:
  - Desired output: r := p + q = P(0) + Q(0)
  - R := P + Q is a valid Shamir polynomial (degree ≤ t-1 and R(0) = r)
  - Party i's share is $R(x_i) = P(x_i) + Q(x_i)$
- Parties can construct their share of the output locally, without any interaction!

Multiplication Gate

- Two inputs shared with Shamir's scheme:
  - Secret p, polynomial P such that p = P(0), shares $P(x_1)$, …, $P(x_n)$
  - Secret q, polynomial Q such that q = Q(0), shares $Q(x_1)$, …, $Q(x_n)$
- Output:
  - Desired output: r := p * q = P(0) * Q(0)
  - R := P * Q satisfies R(0) = r but has degree ≤ 2(t-1), not a valid sharing
  - Goal: find another polynomial R' with R'(0) = r and degree ≤ t

Multiplication Gate – Degree Reduction

Goal: find another polynomial R' with R'(0) = r and degree ≤ t-1

Reducing degree by resharing coefficients:

- Observation: with Lagrange's formula, we have $R(0) = \sum_{i=1}^{2t-1} \lambda_i R(x_i)$
- Each party i can create a Shamir sharing of $R(x_i)$:
  - Choose a degree t-1 polynomial R_i such that $R_i(0) = R(x_i)$
  - Distribute $R_i(x_j)$ to party j

$$R(0) = \sum_{i=1}^{2t-1} \lambda_i \left(\sum_{j=1}^{t} \mu_j R_i(x_j) \right) = \sum_{j=1}^{t} \mu_j \left(\sum_{i=1}^{2t-1} \lambda_i R_i \right) (x_j) \qquad R' := \sum_{i=1}^{2t-1} \lambda_i R_i$$

Properties:

- Re-sharing requires extra communication
- Security against t-1 corrupt parties. We also need 2t-1 ≤ n to reconstruct R(0): **honest majority**.
- Corrupt parties are still semi-honest here (imagine a malicious party that re-shares garbage coefficients)

3. MPC with Preprocessing: Beaver Triples

Beaver, 1991 [10]

- BGW multiplications are costly (in terms of interactions)
- We can save time by computing some things in advance
- MPC with preprocessing:
  - Offline phase: a trusted dealer generates input-independent cryptographic material
  - Online phase: parties use the material to save some time (less communication) when evaluating the circuit
- Beaver triples are secret-shared tuples for multiplication

3. Beaver Triples

Generation:

1. Take a random tuple (a,b,c) in F_p such that c = a*b
2. Split it and distribute shares to the parties: [a], [b], [c]

Multiplication: we have [x], [y] and want [xy]

1. Each party reveals [x] - [a], d := x - a is now public
2. Each party reveals [y] - [b], e: y - b is now public
3. Each party computes locally [xy] = de + d[b] + e[a] + [c]

3. Beaver Triples

Security:

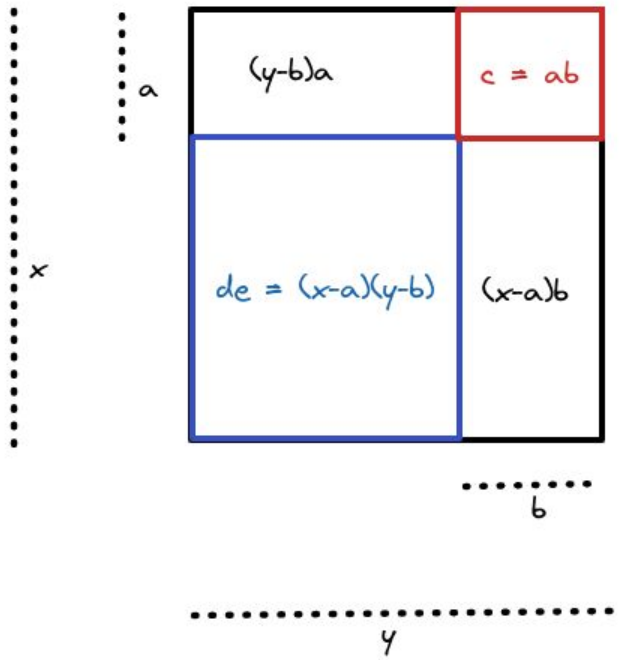
- x - a and y - b are one-time pad encryptions of x and y

Correctness:

$$\sum (de + d[b] + e[a] + [c])$$

$$= (x-a)(y-b) + (x-a)b + (y-b)a + c$$

$$= xy$$



Beaver Triples in a Circuit

Computational and communication cost:

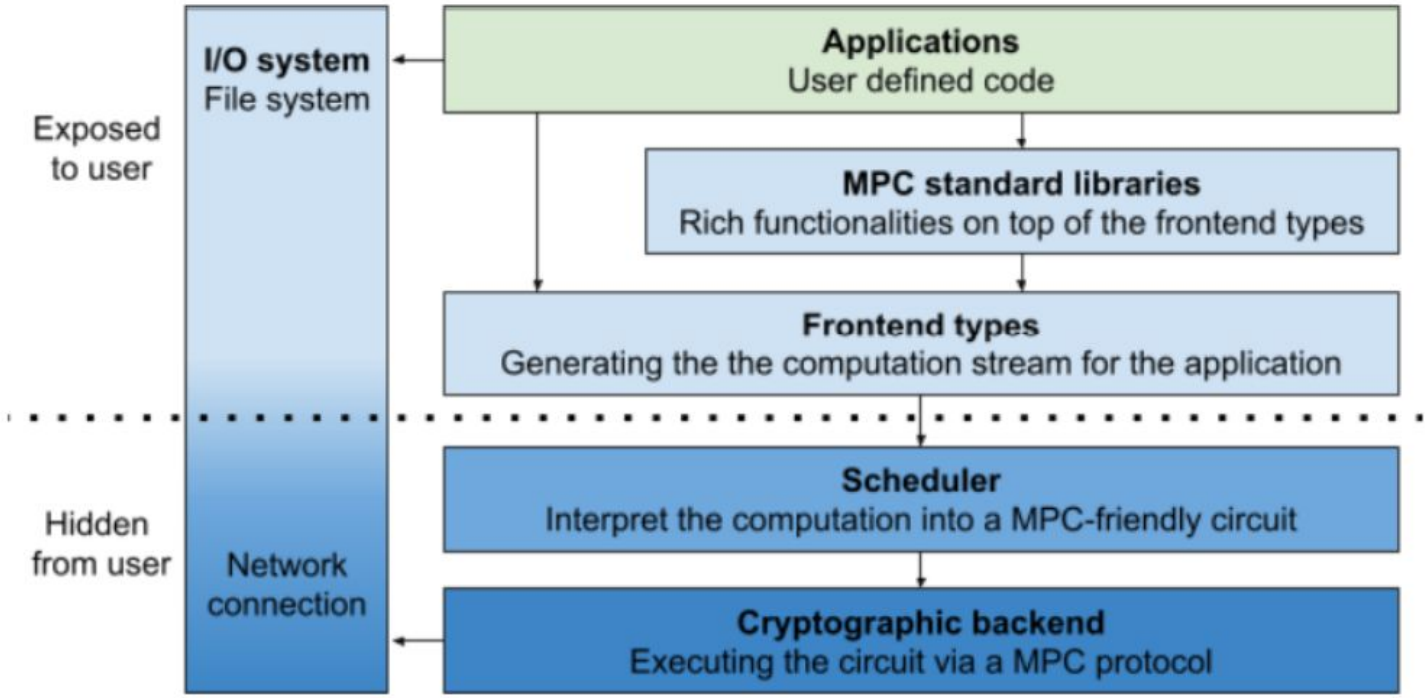
- Each party just needs to broadcast 2 values ([x] - [a] and [y] - [b])
- In BGW, each party generates a polynomial and sends n values (one for each other party)
- Triples don’t depend on the input, and can’t be reused, so we need to prepare enough to evaluate the whole circuit
- There are techniques to generate triples in batches

Applicability:

- Beaver triples work with other types of secret sharing, not just Shamir and BGW
- The trusted dealer can be emulated by the parties themselves (e.g. with HE [3])
- Information-theoretic security: no computational assumptions

4. Implementation: Meta's Private Computation Framework

- General purpose library to build MPC *systems*
- Open-source: https://github.com/facebookresearch/fbpcf
- Architecture from the whitepaper [2]:



Cyptographic backend and scheduler

- Boolean circuits instead of arithmetic circuits
  - Inputs are secret-shared bits
  - AND and XOR instead of + and x
  - Easier to manipulate and compile programs
- Cryptographic primitives:
  - GMW secret sharing, a different scheme than BGW tailored for F_2 and resilient against up to n-1 corrupt parties (BGW needs a honest majority)
  - Beaver triples for AND gates
  - https://github.com/facebookresearch/fbpcf/blob/main/fbpcf/engine/SecretShareEngine.cpp
- Scheduler:
  - Keep track of intermediate results
  - Order gates and execute them
  - Supports multithreading

C++ types and operators

- Frontend types: special C++ types for Bit, Int, BitString
- Everything is reduced to bitwise operations (gates)
- Gates are passed to the scheduler
- Example: integer comparison. https://github.com/facebookresearch/fbpcf/blob/b38024cccc79dff74bbce3fbbf9836caf80a4ce7/fbpcf/frontend/Int_impl.h#L186

Example application

- The millionaire game:
  - Alice and Bob
  - Each party has one (secret) input, corresponding to their wealth
  - The output of the circuit is one bit, corresponding to who is the richest (but not their wealth)
  - Parties shouldn’t learn anything else than the output
- https://github.com/facebookresearch/fbpcf/blob/main/example/millionaire/MillionaireGame.h
- Deployment: TCP socket communication, parties can run in Docker

Conclusion

- Simple setting: honest-but-curious adversary and information-theoretic security
- Basic MPC techniques: Shamir secret sharing, BGW protocol, Beaver triples
- Local computations are lightweight (unlike FHE)
- But parties need to communicate more often

Conclusion

There are many other important concepts we didn't cover. Some keywords:

- **Malicious security**: we can adapt protocols with MACs, ZK proofs and other techniques (e.g. see the SPDZ family of protocols and its modern implementations [4]).
- **Oblivious transfer (OT)**: a useful primitive where a receiver privately picks one of two secrets offered by a sender.
- **Garbled circuits**: evaluate circuits in constant number of rounds (BGW's number of rounds is proportional to the depth of the circuit).
- **FHE and Homomorphic Secret Sharing**: other ways of achieving MPC.
- **Oblivious RAM (ORAM):** hide data access patterns efficiently.

Conclusion

State-of-the-art MPC protocols can be practical:

- Usually with 2 or 3 *active* parties (think non-colluding cloud providers)
- But can handle large numbers of *passives* parties (think browsers) who share their input once and let the active parties compute the output
- Primitives tailored for different use cases

Examples:

- AES evaluation on a secret-shared secret key [5]
- Distributed aggregation for contact tracing or telemetry [7]
- Training ML models on secret-shared data [6]

References

[1] D. Evans, V. Kolesnikov, and M. Rosulek, “A Pragmatic Introduction to Secure Multi-Party Computation,” SEC, vol. 2, no. 2–3, pp. 70–246, Dec. 2018, doi: 10.1561/3300000019.

[2] “Private Computation Framework 2.0 - Meta Research,” Meta Research. https://research.facebook.com/publications/private-computation-framework-2-0/ (accessed Mar. 08, 2023).

[3] N. P. Smart and T. Tanguy, “TaaS: Commodity MPC via Triples-as-a-Service,” in Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, New York, NY, USA, Nov. 2019, pp. 105–116. doi: 10.1145/3338466.3358918.

[4] M. Keller, “MP-SPDZ: A Versatile Framework for Multi-Party Computation,” in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, Nov. 2020, pp. 1575–1590. doi: 10.1145/3372297.3417872.

[5] I. Damgård and M. Keller, “Secure Multiparty AES: (Short Paper),” in Financial Cryptography and Data Security, vol. 6052, R. Sion, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 367–374. doi: 10.1007/978-3-642-14577-3_31.

[6] P. Mohassel and P. Rindal, “ABY3: A Mixed Protocol Framework for Machine Learning,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, Oct. 2018, pp. 35–52. doi: 10.1145/3243734.3243760.

[7] H. Corrigan-Gibbs and D. Boneh, “Prio: Private, Robust, and Scalable Computation of Aggregate Statistics,” presented at the 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), 2017, pp. 259–282. Accessed: Dec. 15, 2020. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs

[8] A. Shamir, “How to share a secret,” Commun. ACM, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.

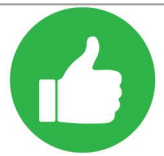
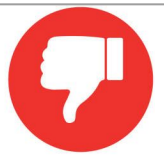
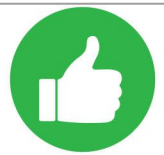
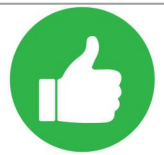
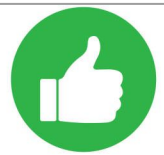
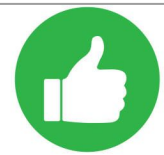
[9] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in Proceedings of the twentieth annual ACM symposium on Theory of computing, New York, NY, USA, Jan. 1988, pp. 1–10. doi: 10.1145/62212.62213.

[10] D. Beaver, “Efficient Multiparty Protocols Using Circuit Randomization,” in Advances in Cryptology — CRYPTO ’91, vol. 576, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432. doi: 10.1007/3-540-46766-1_34.

The End

Connections and Tradeoffs of Advanced Privacy Technologies

Threats and Tradeoffs of Privacy in ML

Privacy Tech	Threat	Strength of guarantee	Performance impact	Accuracy impact
Differential privacy	leakage of training data through models			
Homomorphic encryption	untrusted cloud's access to data during computation			
Hardware enclaves	untrusted cloud's access to data during computation			
Secure multi-party computation	untrusted cloud's access to data during computation			
Federated learning	untrusted cloud's access to data during computation			

Combinations Needed

- DP and the others address orthogonal threats, so for fuller protection, DP should be combined with all others
- Hardware enclaves can speed up homomorphic encryption and secure multi-party computation
- Federated learning has weak privacy, but can be combined with DP for strong privacy, with some loss in accuracy

Broader Connections

- Connections exist between privacy and other desirable properties of ML
- In theory, this could mean that technologies for one property could be useful for other properties
- Practical approaches to exploit these connections are still being researched

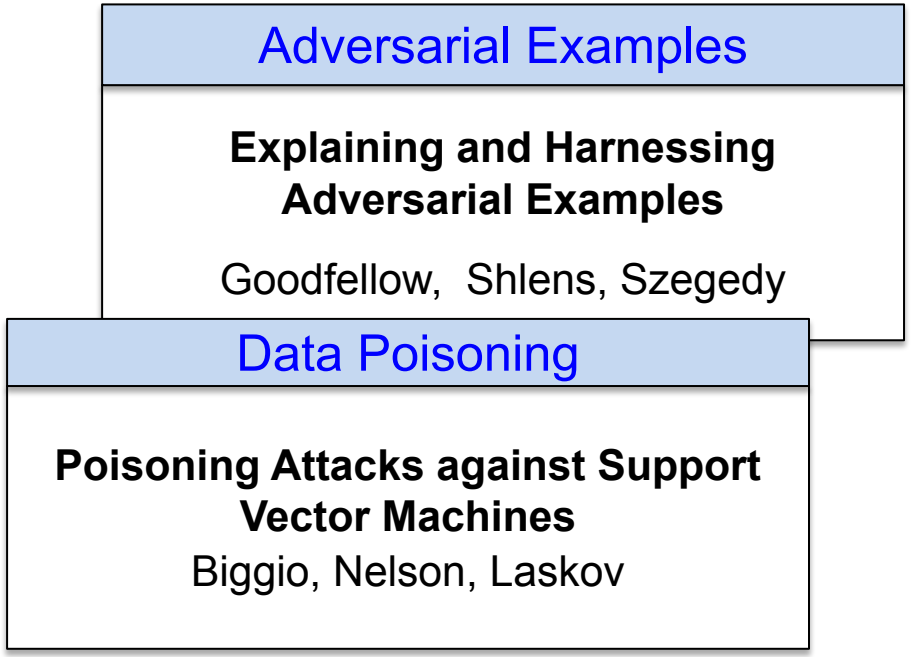
(NOTE: We started talking about these in the DP lecture, but we rushed and didn't go into any details and all connections. We will discuss those today, but note that the slides are identical.)

Myriad of ML Concerns

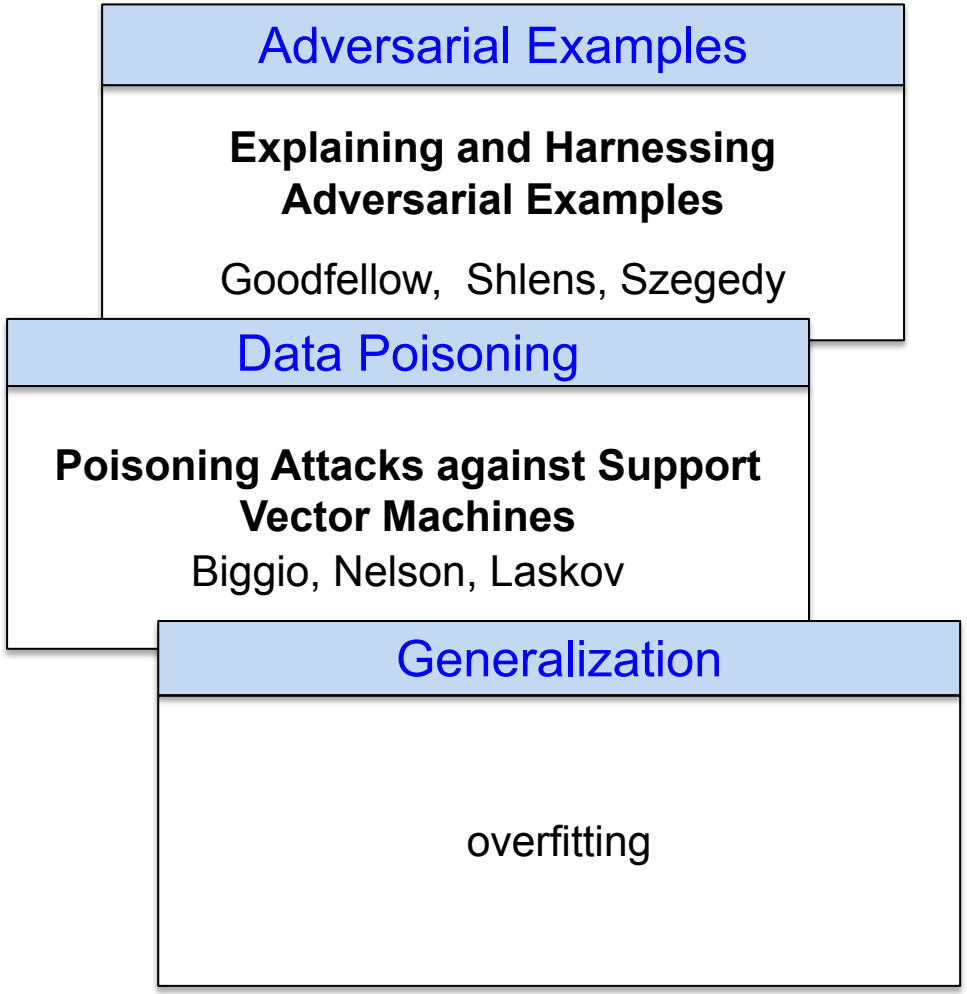
Myriad of ML Concerns

Adversarial Examples
Explaining and Harnessing Adversarial Examples Goodfellow, Shlens, Szegedy

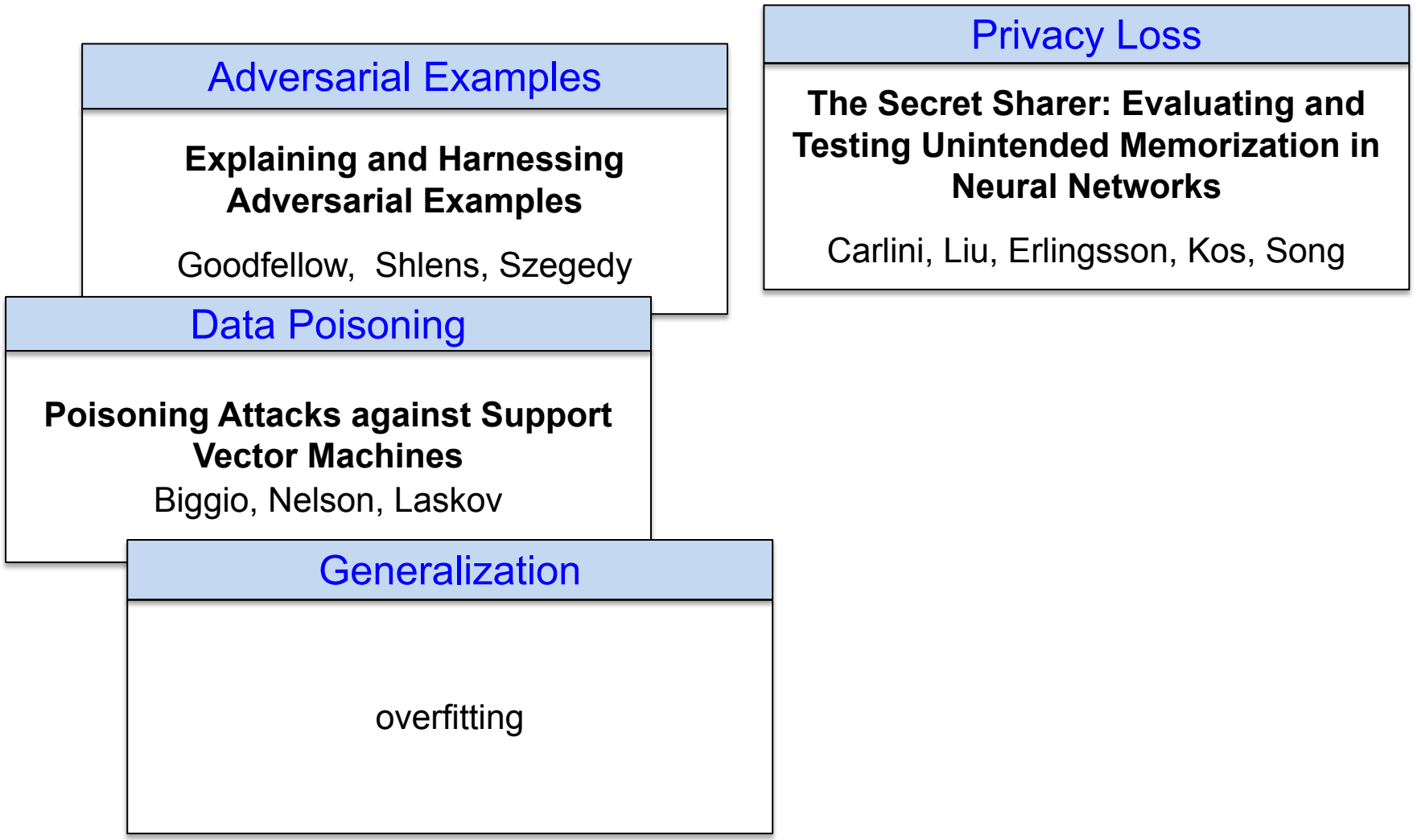
Myriad of ML Concerns



Myriad of ML Concerns



Myriad of ML Concerns



Myriad of ML Concerns

Adversarial Examples

Explaining and Harnessing Adversarial Examples

Goodfellow, Shlens, Szegedy

Privacy Loss

The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks

Carlini, Liu, Erlingsson, Kos, Song

Data Poisoning

Poisoning Attacks against Support Vector Machines

Biggio, Nelson, Laskov

Bias, Discrimination

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Bolukbasi, Chang, Zou, Saligrama, Kalai

Generalization

overfitting

Myriad of ML Concerns

Adversarial Examples

Explaining and Harnessing Adversarial Examples

Goodfellow, Shlens, Szegedy

Privacy Loss

The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks

Carlini, Liu, Erlingsson, Kos, Song

Data Poisoning

Poisoning Attacks against Support Vector Machines

Biggio, Nelson, Laskov

Bias, Discrimination

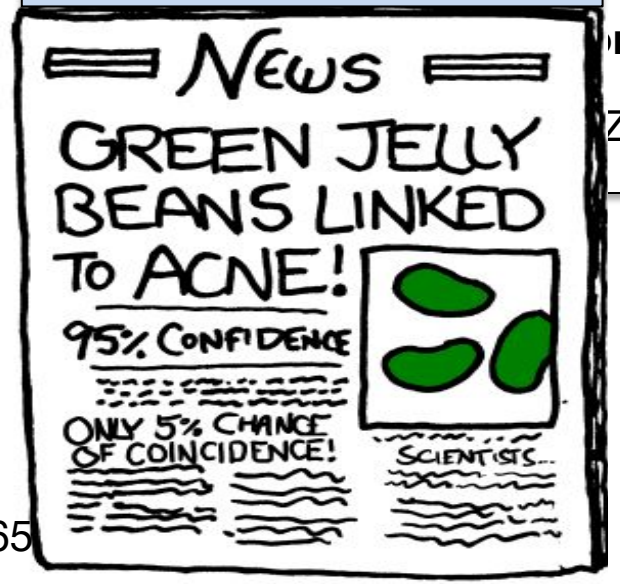
ter Programmer as Homemaker?
rd Embeddings

Zou, Saligrama, Kalai

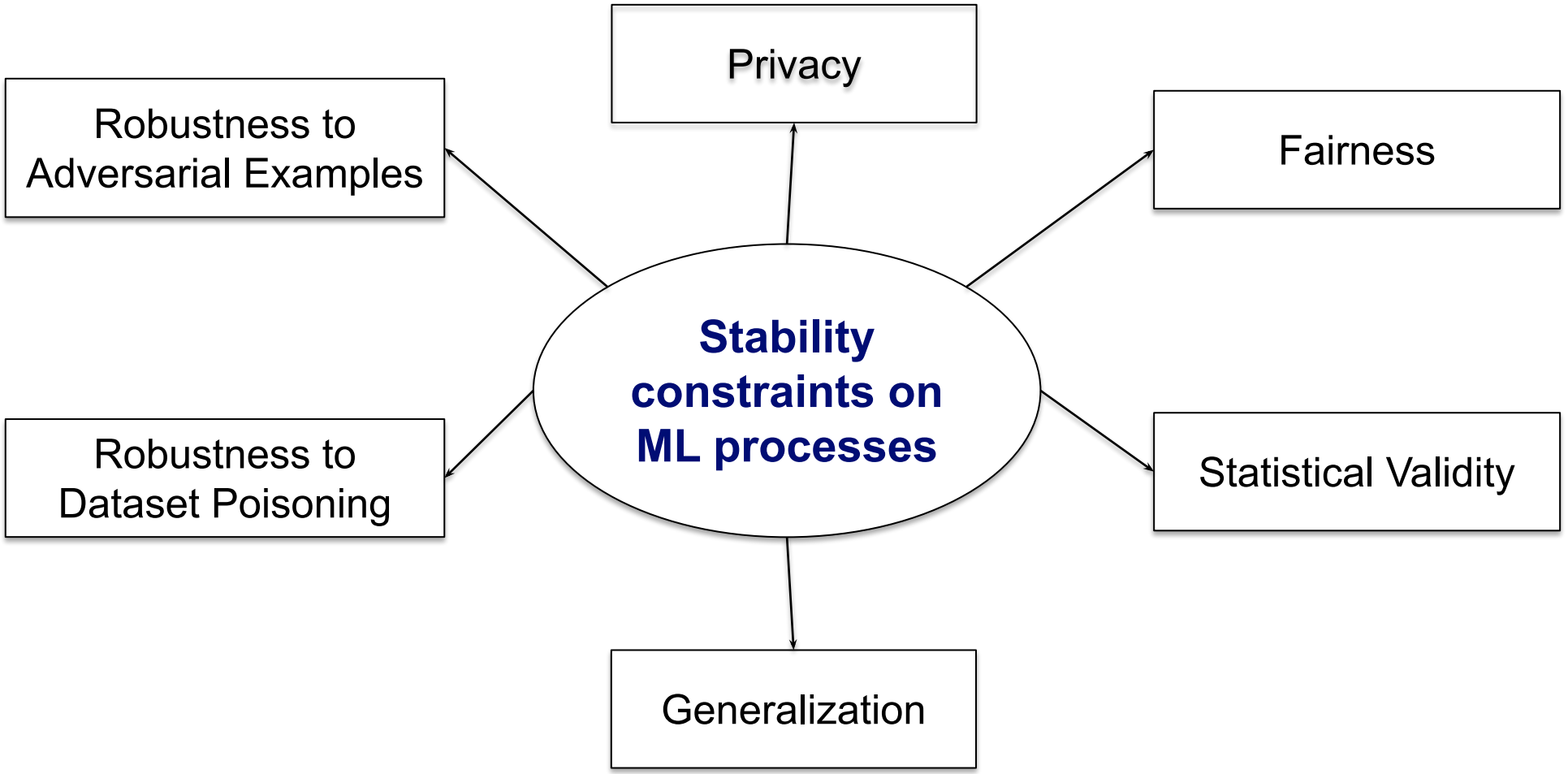
Generalization

overfitting

False Discoveries



Many Concerns Are Related



[Hardt-16]

Example: DP Improves More than Privacy

- DP is a **strong stability constraint** on computations running on datasets: it requires that no single data point in an input dataset has significant influence over the output

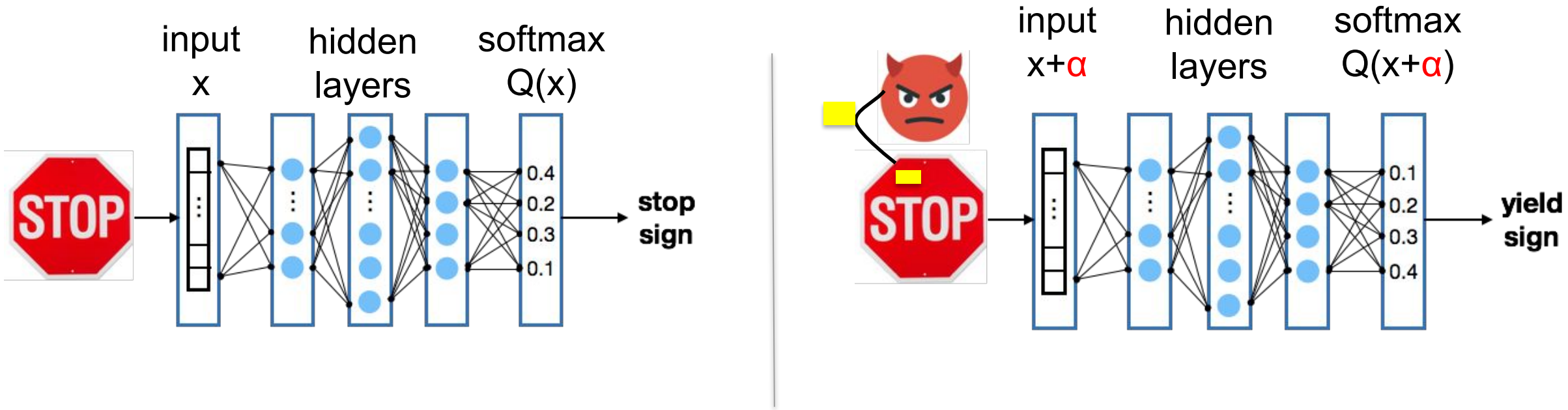
- It has been been shown to improve a variety of desirable ML properties beyond privacy, e.g.:
  - DP for Adversarial Robustness (Lecuyer+19)
  - DP for Generalization (Hardt-16, Bassily+16)
  - DP for Fairness (Dwork+13)
  - DP for Statistical Validity (Dwork+15)

DP for Adversarial Robustness

(Lecuyer+19)

Adversarial Examples

- Adversary finds a tiny perturbation to a correctly classified input that causes misclassification

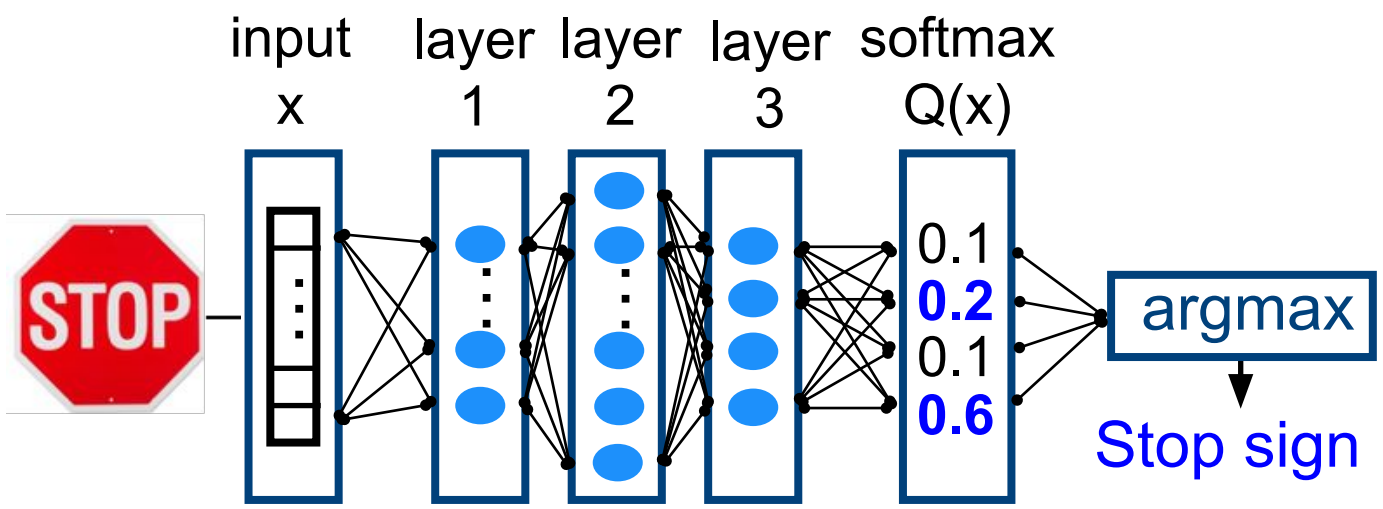


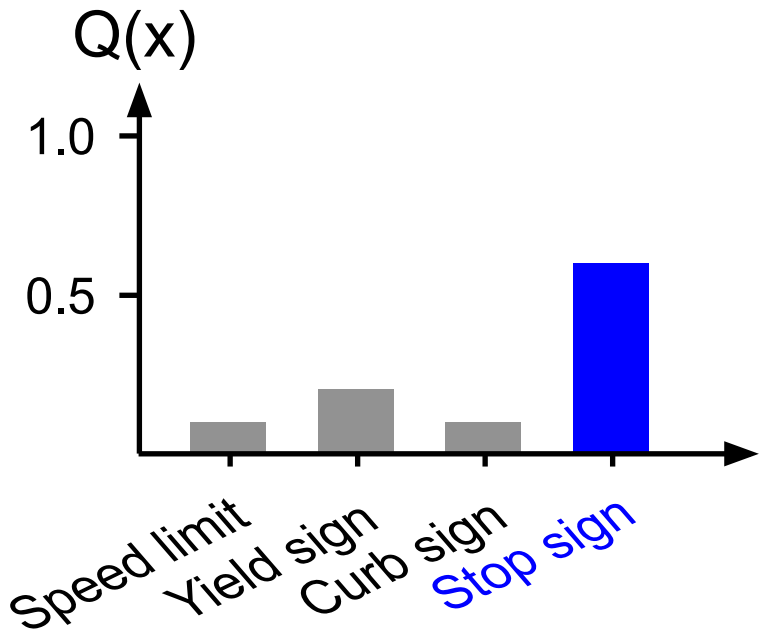
DP for Adversarial Examples

- Problem: small input changes create large score changes
- Approach: **make prediction function DP**

DP for Adversarial Examples

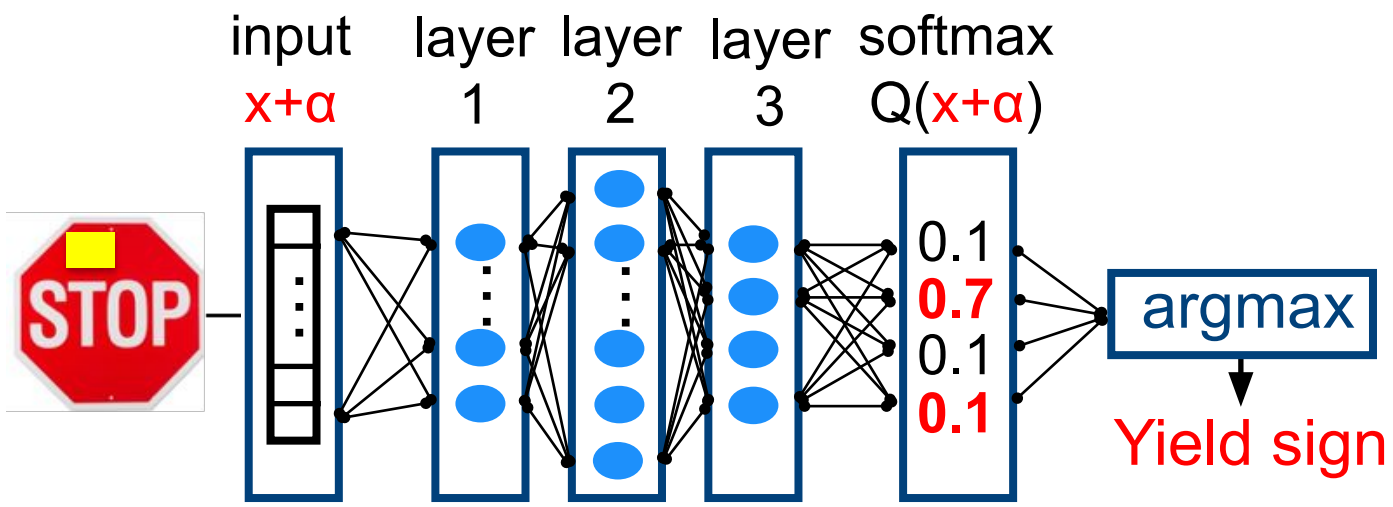
- Problem: small input changes create large score changes
- Approach: **make prediction function DP**

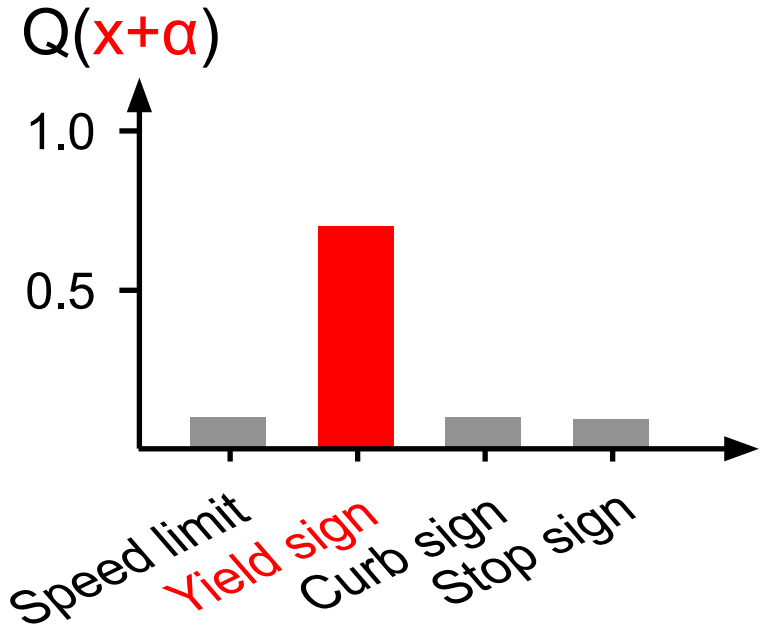




DP for Adversarial Examples

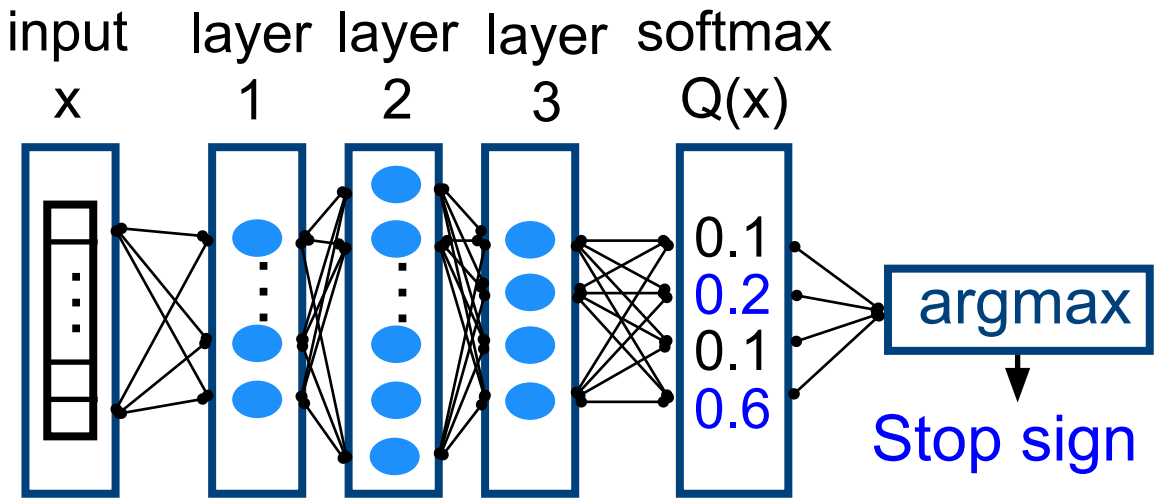
- Problem: small input changes create large score changes
- Approach: **make prediction function DP**





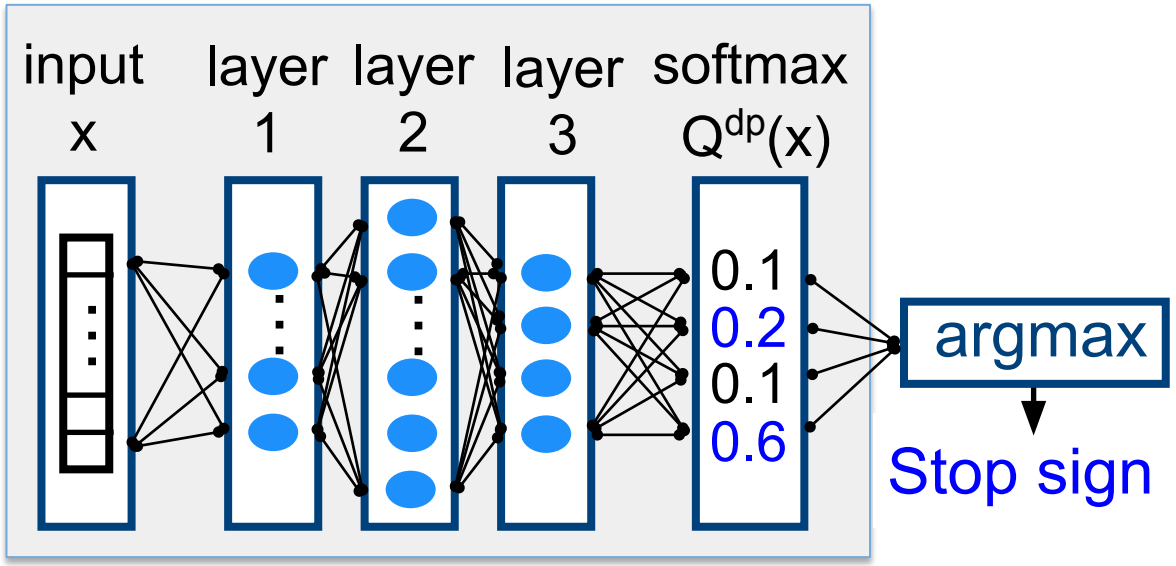
How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



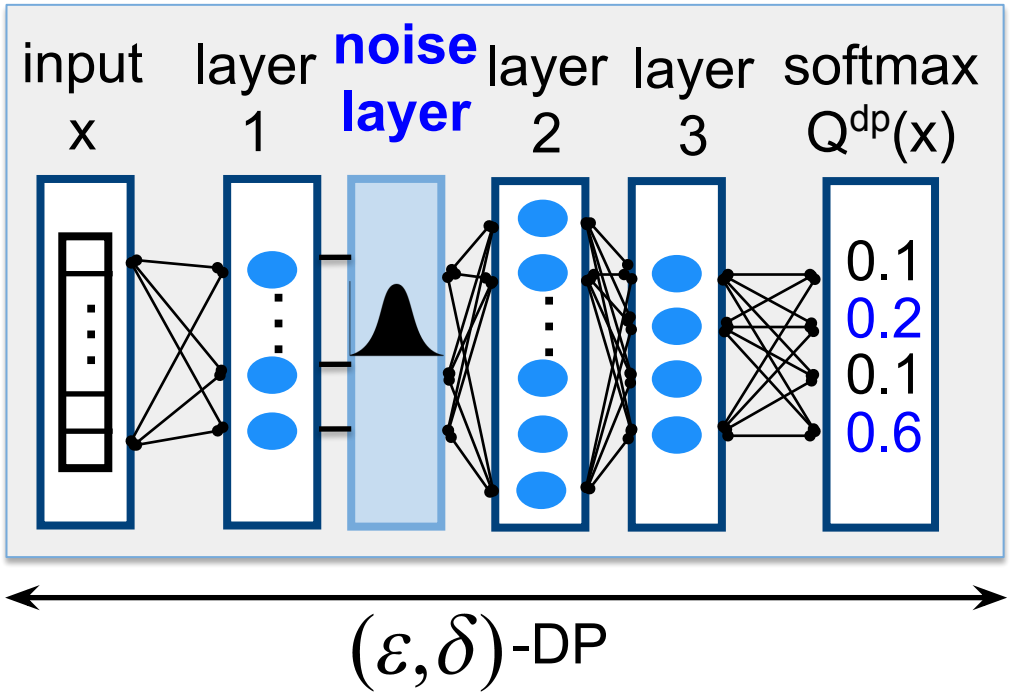
How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



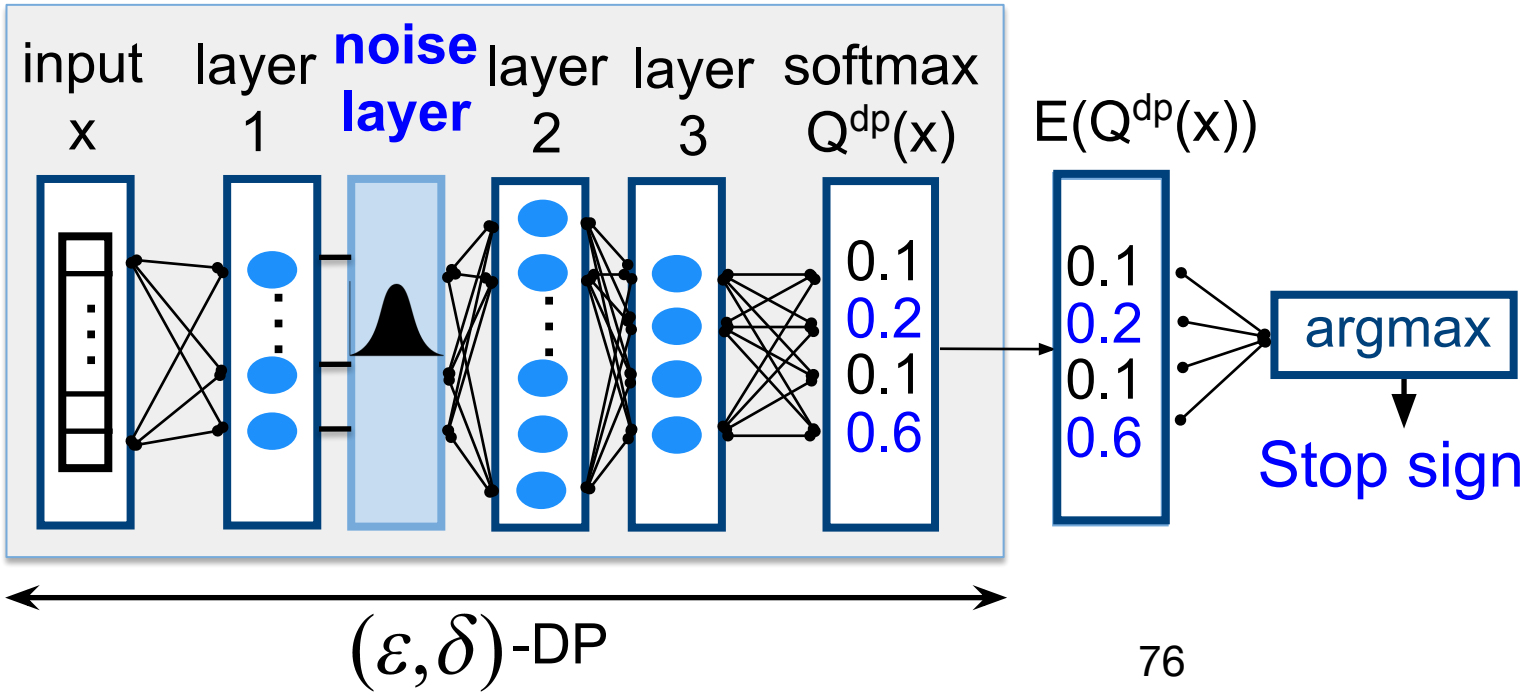
How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



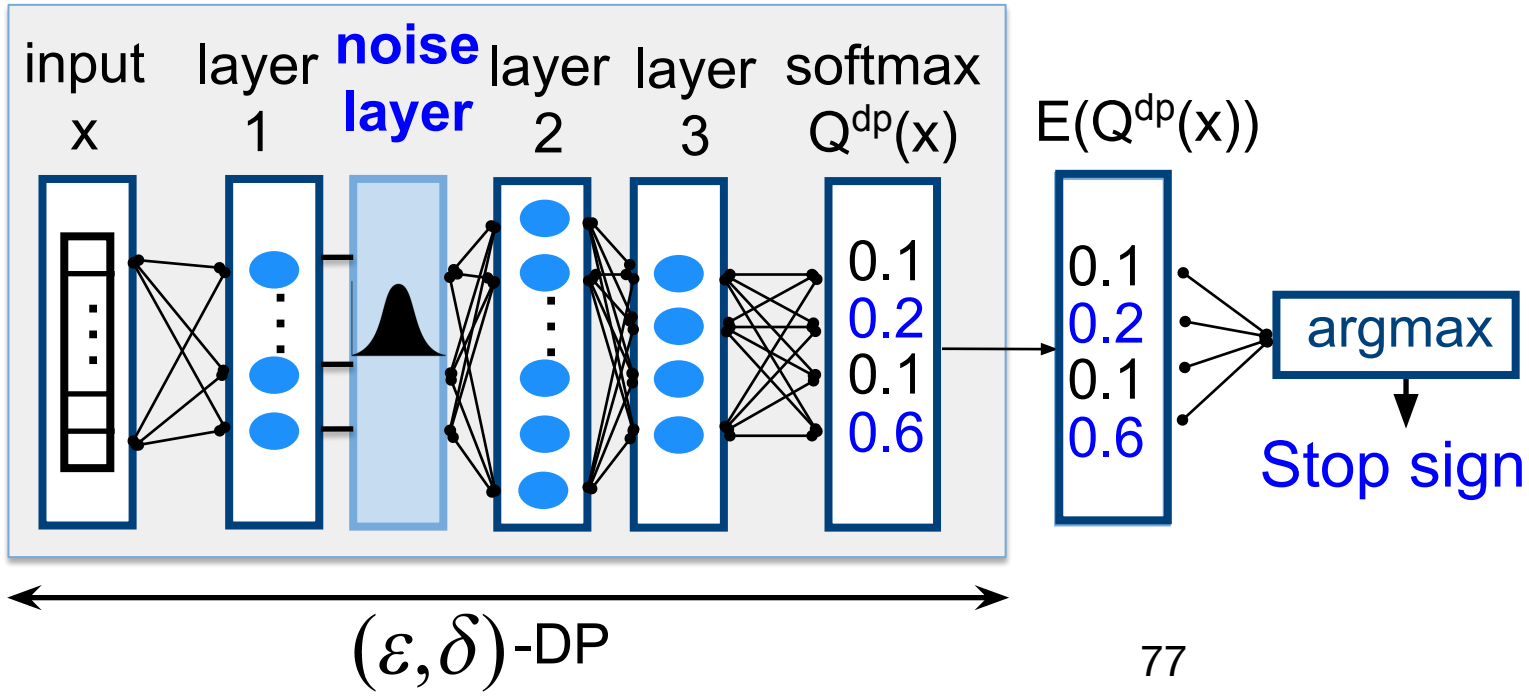
How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



DP's stability bounds on expected scores:

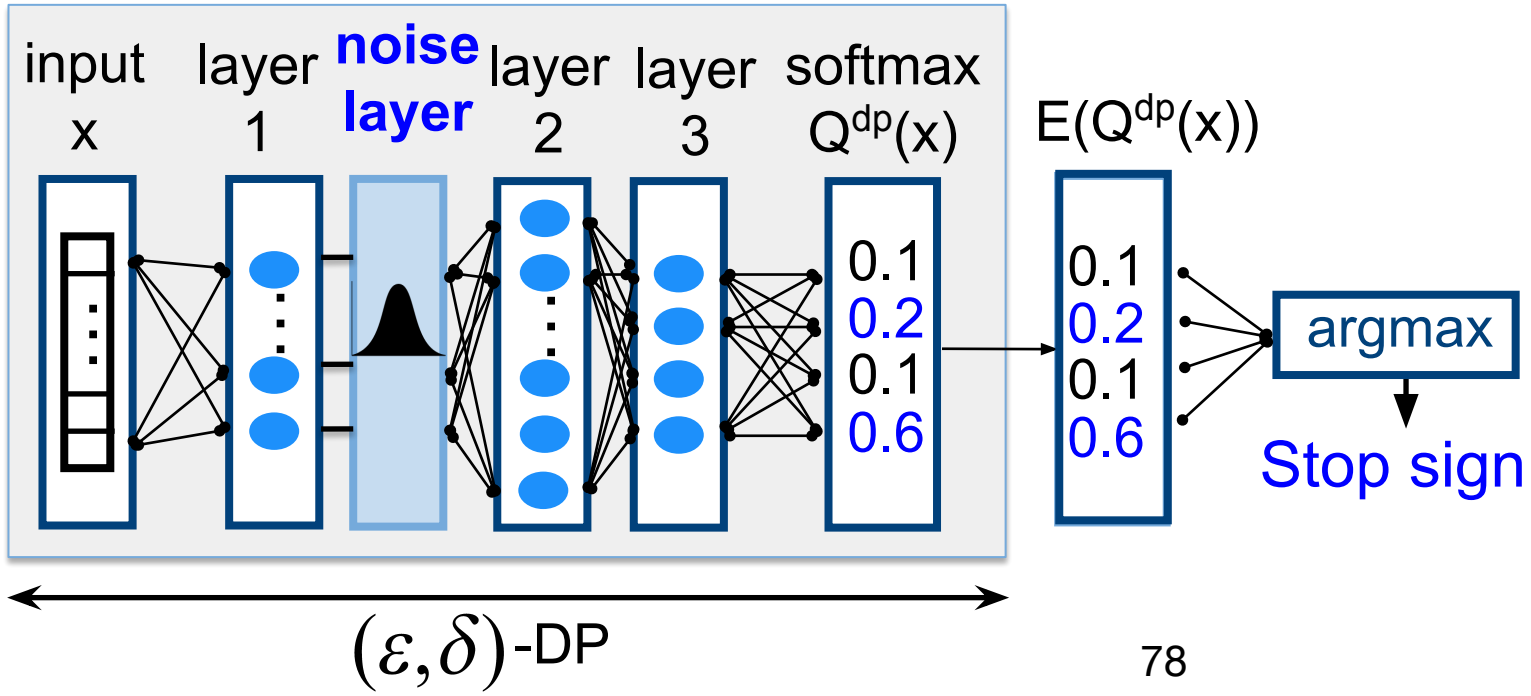
$$\text{for all } \alpha \text{ with } \|\alpha\|_2 = \sqrt{\sum \alpha_i^2} \le L:$$

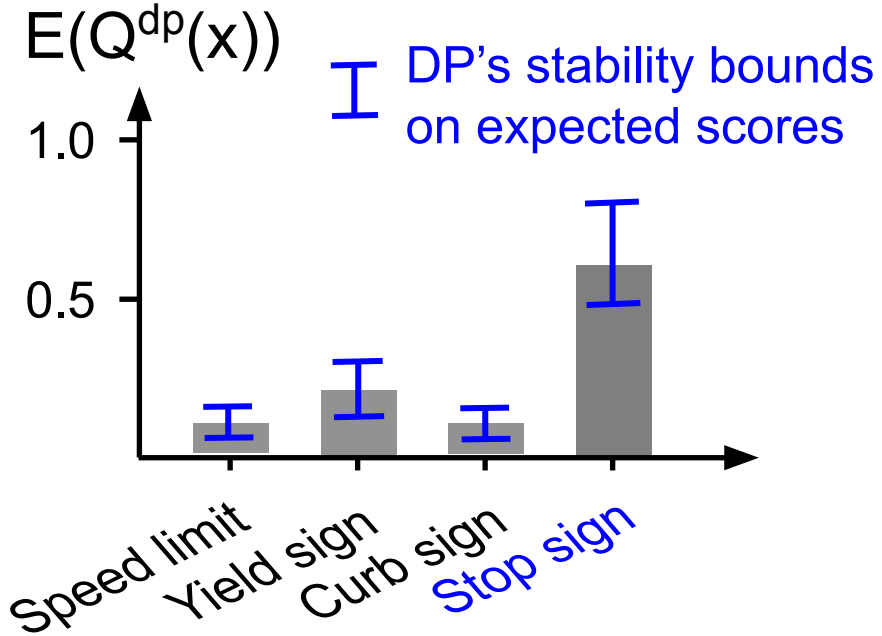
$$\frac{E(Q^{dp}(x)) - \delta}{e^{\varepsilon}} \le E(Q^{dp}(x+\alpha))$$

$$E(Q^{dp}(x+\alpha)) \le e^{\varepsilon} E(Q^{dp}(x)) + \delta$$

How It Works

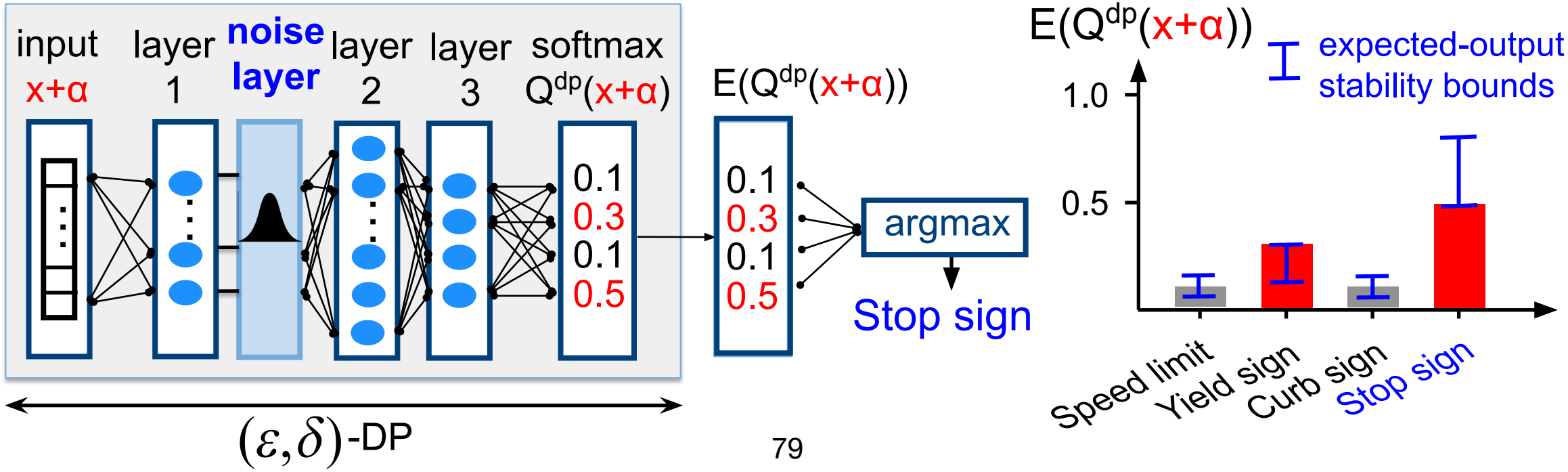
1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x





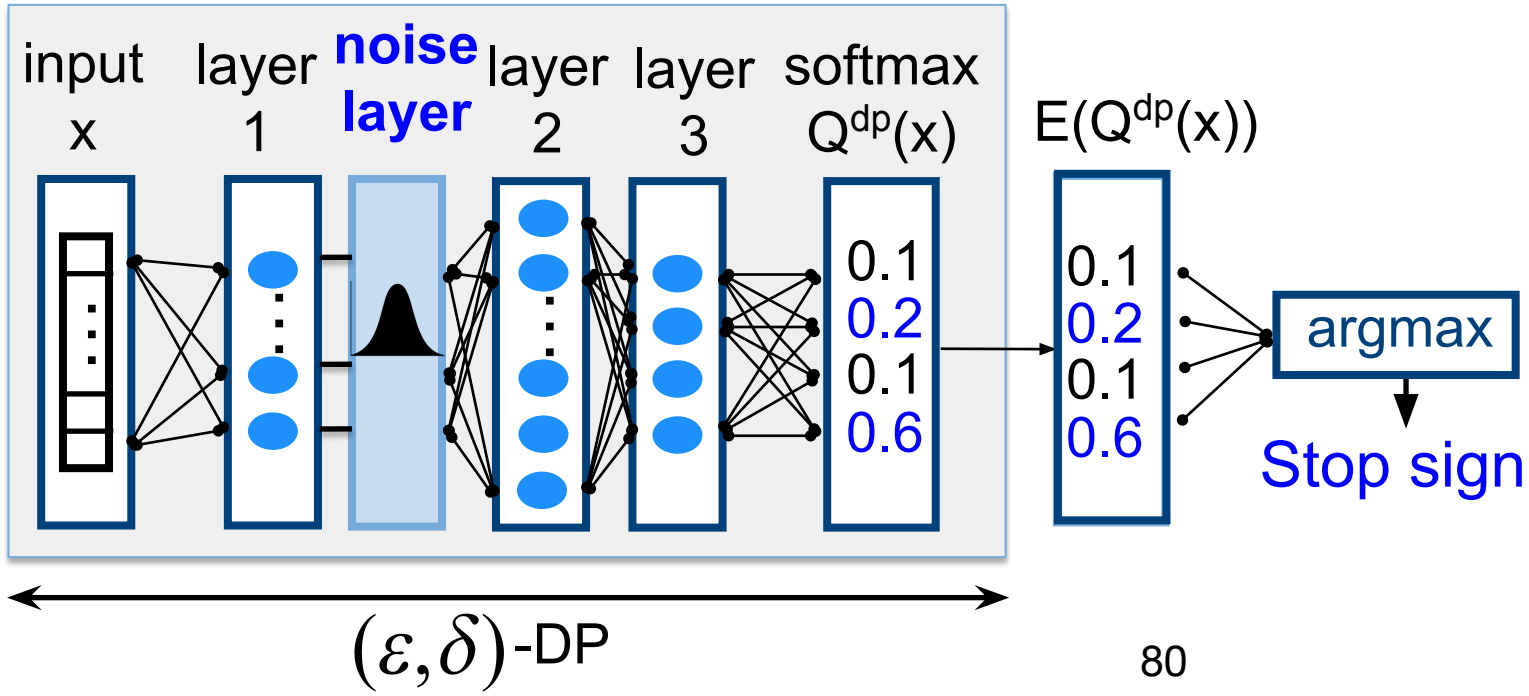
How It Works

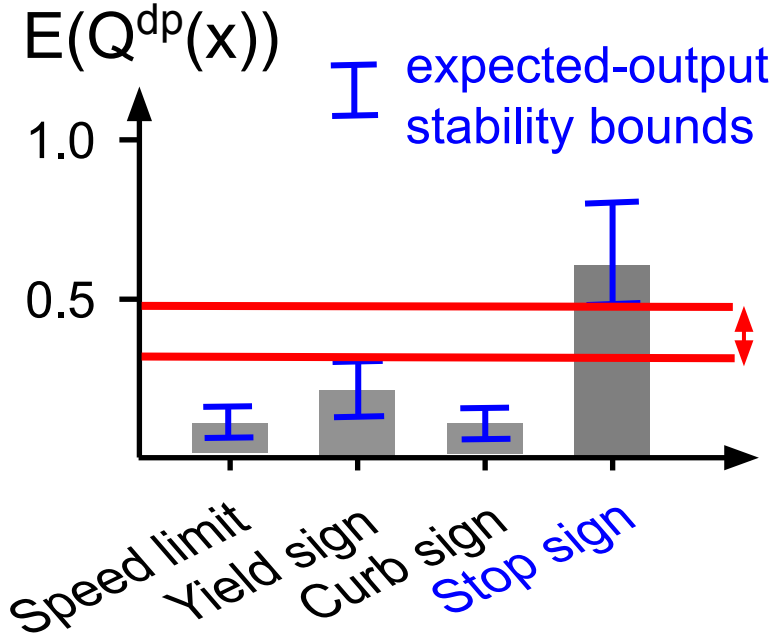
1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x



How It Works

1. Randomize prediction function to make it DP
2. Use expected scores to choose argmax
3. Use DP's stability bounds on expected scores to certify prediction on x





DP for Generalization

(Hardt-16)

Generalization

- Central to ML is our ability to relate how a learning algorithm fares on a sample set to its performance on unseen instances. This is called **generalization**

Generalization

- Central to ML is our ability to relate how a learning algorithm fares on a sample set to its performance on unseen instances. This is called **generalization**

Risk (Out-of-sample Error)

$$R = \mathop{\mathrm{E}}_{z \sim D}\left[\ell(A(S), z)\right]$$

Empirical Risk (Train Error)

$$R_S = \frac{1}{n}\sum_{i=1}^{n}\ell(A(S), s_i)$$

Generalization Error

$$R - R_S$$

A= training function; D= input distribution; S= training set; n=|S|; ℓ = loss function

Generalization

- Central to ML is our ability to relate how a learning algorithm fares on a sample set to its performance on unseen instances. This is called **generalization**

Risk (Out-of-sample Error)

$$R = \mathop{\mathrm{E}}_{z \sim D}\left[\ell(A(S), z)\right]$$

Empirical Risk (Train Error)

$$R_S = \frac{1}{n}\sum_{i=1}^{n}\ell(A(S), s_i)$$

Generalization Error

$$R - R_S$$

A= training function; D= input distribution; S= training set; n=|S|; ℓ = loss function

- We care about R. If we manage to minimize R_S, all that matters is the **generalization error**. Many approaches exist that improve generalization error (mostly statistical)

Generalization ⇔ Stability

- **Thm: In expectation, generalization equals stability**
  - Proof in (Hardt-16)
- An algorithm is **stable** if its output doesn't change much if we perturb the input sample in a single point
- The theorem says that stability is **necessary and sufficient** for generalization

DP for Generalization

- DP is a strong stability constraint on algorithms
- DP thus provides an algorithmic approach to generalization in ML: **make the training function DP**
- It's been long known that adding randomness into training improves generalization
- The level of randomness added is likely insufficient to offer meaningful privacy, but the link DP<->generalization suggests that privacy isn't fundamentally at odds with functionality in ML

DP for Fairness

(Dwork+13)

Individual Fairness

- People who are similar from the perspective of the task at hand should be treated similarly
  - E.g., people with similar capabilities w.r.t. to a graduate program should all be either admitted or rejected
- But in ML, because of data biases and algorithmic amplification of them, small changes in people's relevant capabilities can lead to large changes in the predictions
- That's a sign of instability of the prediction function

88

DP for Individual Fairness

- Approach: **make the prediction function DP**
  - Similar to PixelDP, apply extension of DP to a distance metric among people with respect to their abilities for a task

- While in theory interesting, this approach is not very practical because it relies on a good distance metric among people, which is hard to define

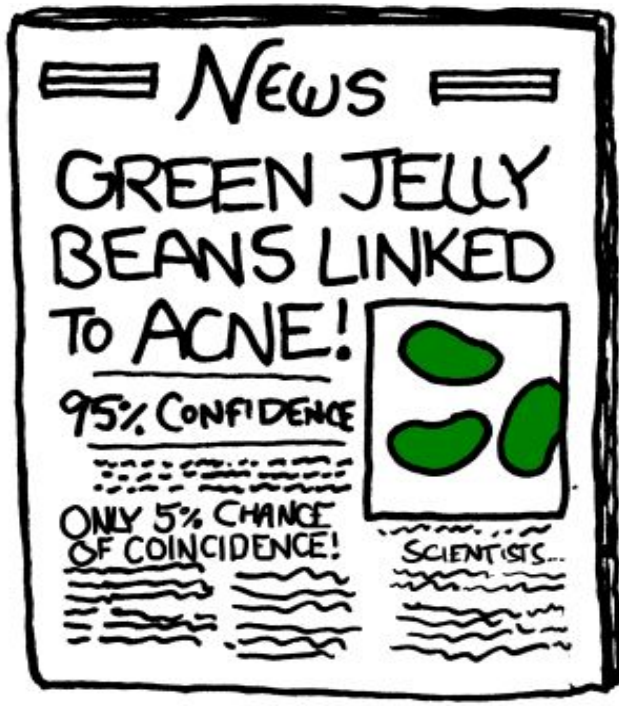
89

DP for Statistical Validity

(Dwork+15)

False Discoveries

- **Ideal scientific method:** Formulate your hypothesis, design your experiment to collect data, test your hypothesis on the data, report finding if statistically significant, and throw away the data.
- **In reality:** data is collected and reused to refine hypotheses, and the new hypotheses are tested on the same data, multiple times.
- Adaptive data reuse breaks assumptions of independence between hypotheses and test data, which hypothesis tests make to ensure statistical validity of the results. Referred to as p-hacking.



A Baseline Approach

- A baseline approach to allow statistical validity on top of a dataset collected from one study is to **split the dataset** into k components, where k is the number of hypotheses you anticipate testing on that dataset adaptively
- Each hypothesis runs on n/k points, so you can only run **k<<n adaptive hypothesis tests** on a dataset of size n
- Can we do better?

92

DP for Statistical Validity

- Problem: you're learning too much from the dataset, therefore your conclusions may overfit it and inherit its biases
- Approach: **make hypothesis tests DP and run on entire dataset**
- Recall DP supports adaptive composition. If you formulate a new hypothesis based on the results of a DP statistical test, and then you test again on the same dataset, you still have a bound on how much information you've extracted from your observations
- You can thus bound the number of tests you can perform while maintaining statistical validity. With advanced composition, the number of adaptive tests you can afford to run is **O(n^2)**

Take-Aways

- Many challenges in ML can be attributed to instability of some algorithm involved in learning: training, prediction, testing

- DP is a very strong stability constraint on algorithms. It thus has broad connections with many desirable properties in ML:
  - Training set privacy: make training function DP
  - Adversarial robustness: make prediction function DP
  - Generalization: make training function DP
  - Fairness: make prediction function DP
  - Statistical validity: make hypothesis test or model evaluation DP

- However, DP may be overly strong for some of these, and that impacts accuracy! Balance is needed, and future research may provide that

Cited References

(Bassily+16) R. Bassily, K. Nissim, A. Smith, T. Steinke, U. Stemmer, and J. Ullman. *Algorithmic stability for adaptive data analysis*. STOC 2016

(Dwork+15) C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, A. Roth. *Preserving Statistical Validity in Adaptive Data Analysis*. STOC 2015

(Hardt-16) M. Hardt. *Stability as a foundation for machine learning.* Blog post, 2016

(Lecuyer+19) M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, S. Jana. *Certified Robustness to Adversarial Examples with Differential Privacy*. IEEE Security & Privacy, 2019

Cited References

(Vadhan, 2016) Vadhan. *The complexity of differential privacy.* https://privacytools.seas.harvard.edu/files/privacytools/files/complexityprivacy_1.pdf.

The End