Computing on Private Data:
Private Genomics and More

David Wu
UT Austin

Computing on Private Data

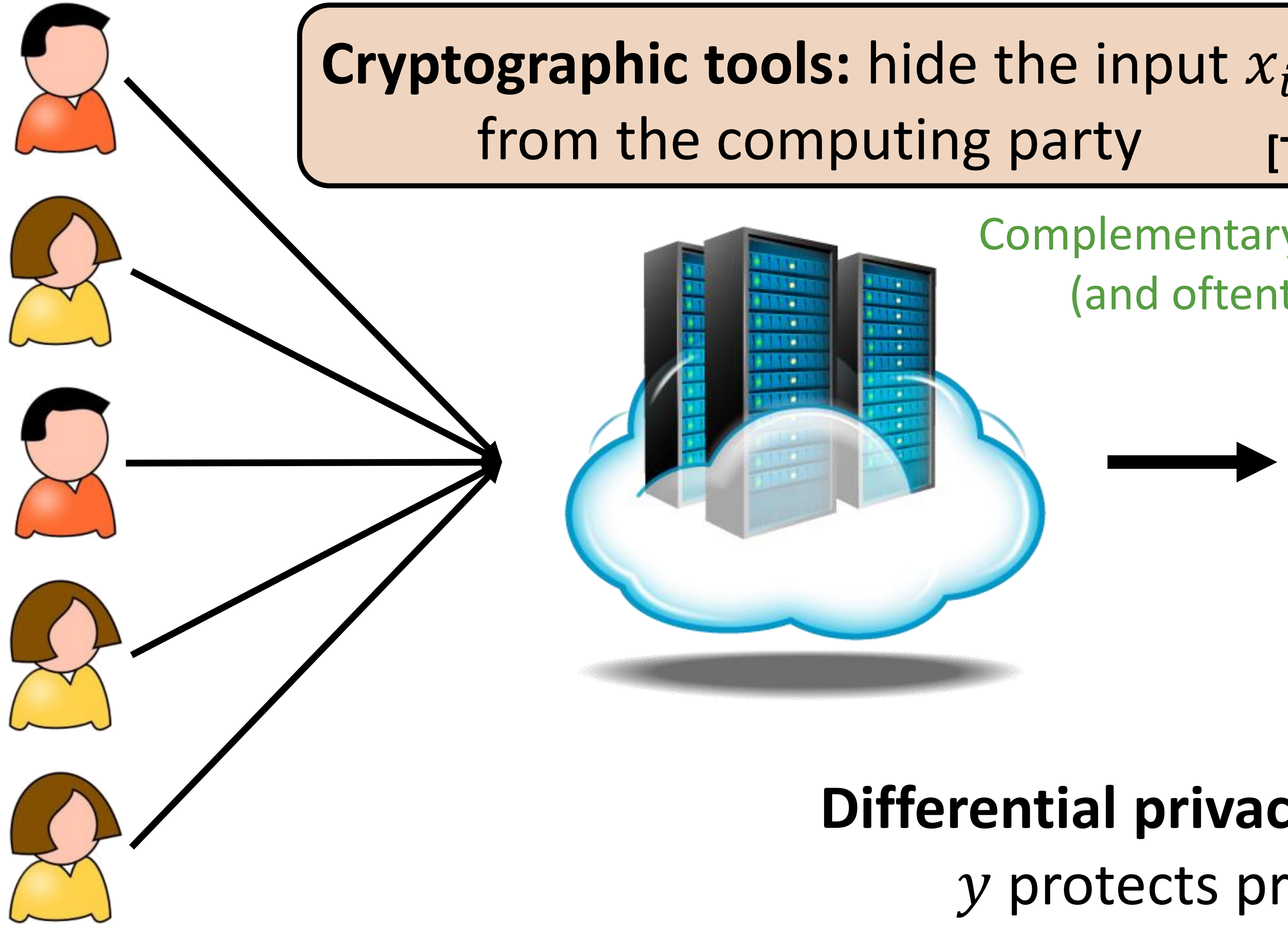
x_1

x_2

x_3

x_4

x_5



Cryptographic tools: hide the input x_i from the computing party **[This talk]**

Complementary goals as differential privacy (and oftentimes, can be *combined*)

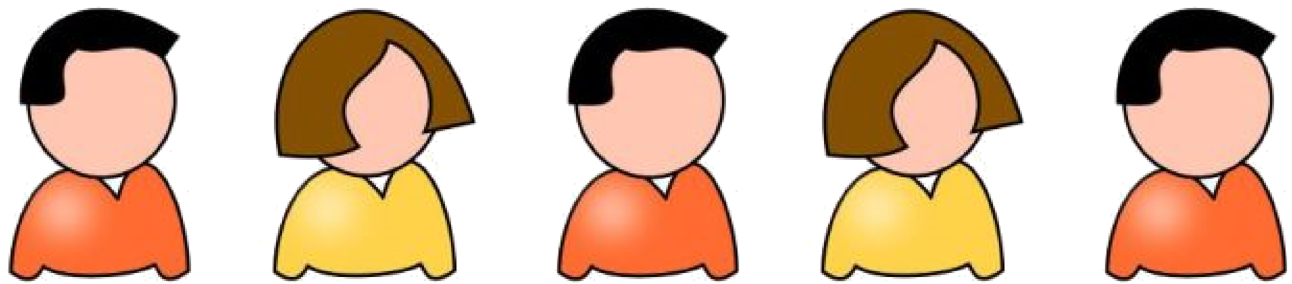
Aggregate statistic
$y \leftarrow f(x_1, \ldots, x_5)$

Differential privacy: ensure that output y protects privacy of input x_i

Rare Disease Diagnosis

[JWBBB17]

What gene causes a specific (rare) disease?



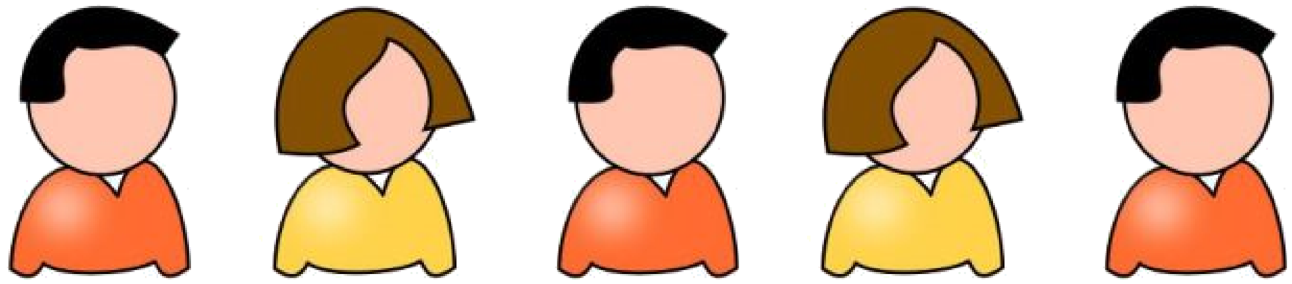
Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Rare Disease Diagnosis

[JWBBB17]

Gene					
A1BG	0	1	0	0	0
	1	1	1	0	1
	⋮	⋮	⋮	⋮	⋮
ZZZ3	0	0	1	0	0



Patients with Kabuki Syndrome

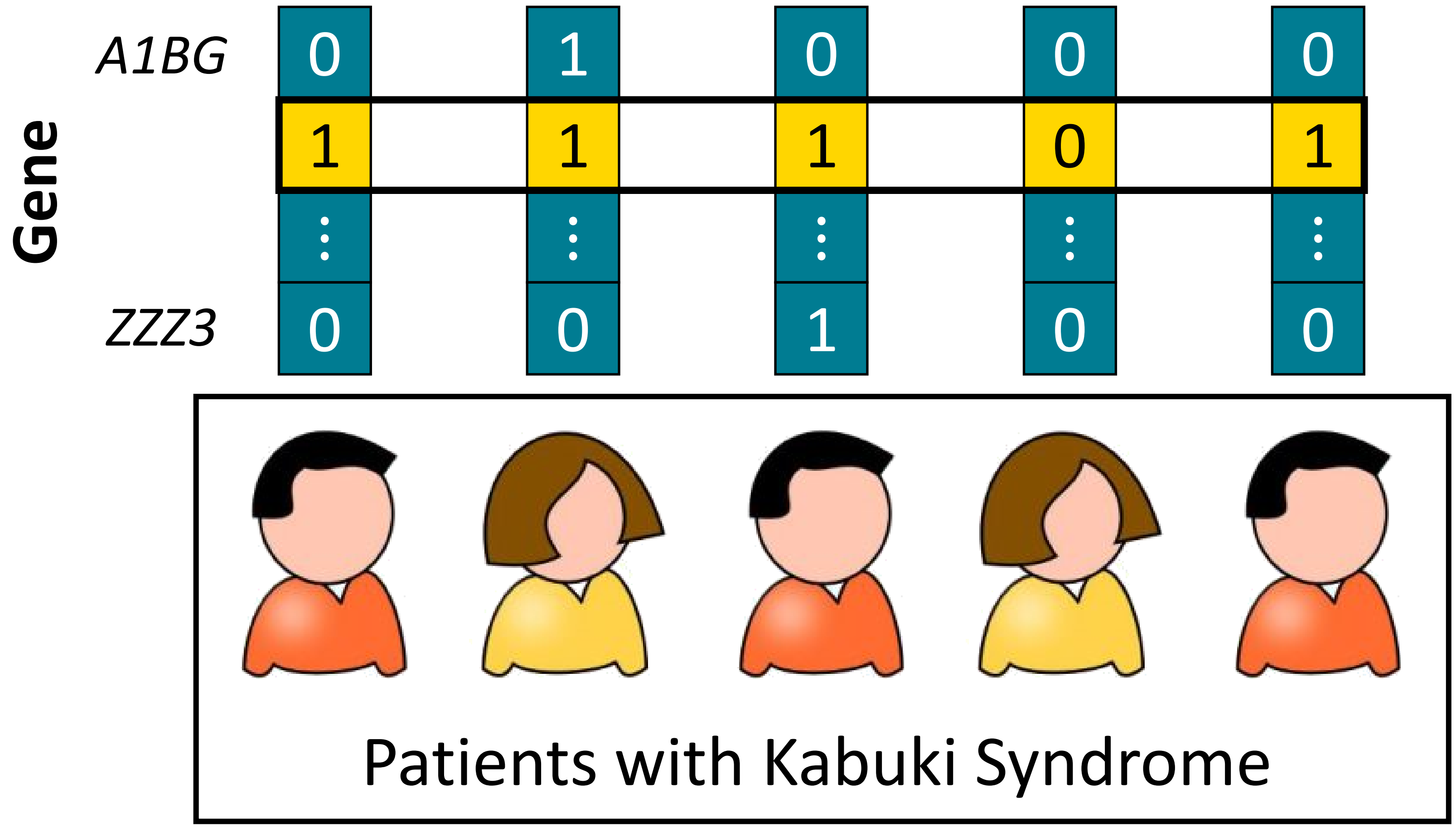
Each patient has a list of 200-400 rare variants over ≈20,000 genes

Each patient has a vector v where $v_i = 1$ if patient has a rare variant in gene i

Goal: Identify gene with most variants across all patients

Rare Disease Diagnosis

[JWBBB17]



Each patient has a list of 200-400 rare variants over ≈20,000 genes

Each patient has a vector v where $v_i = 1$ if patient has a rare variant in gene i

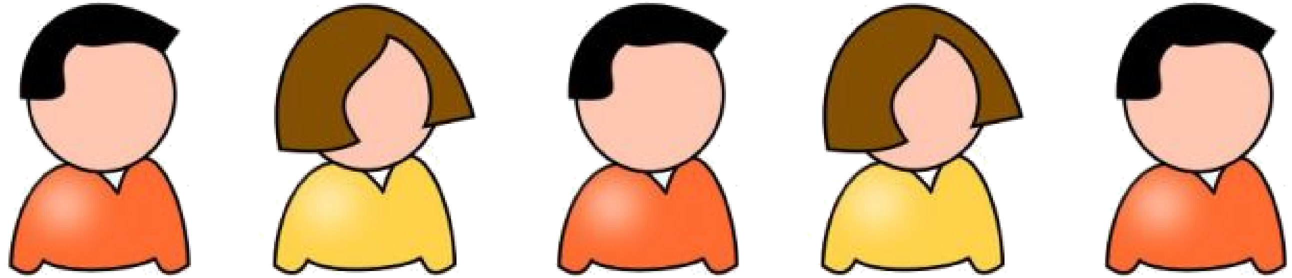
Goal: Identify gene with most variants across all patients

Works well for <u>Mendelian</u> (monogenic) diseases (estimated to affect ≈10% of individuals)

Rare Disease Diagnosis

[JWBBB17]

Gene					
A1BG	0	1	0	0	0
	1	1	1	0	1
	⋮	⋮	⋮	⋮	⋮
ZZZ3	0	0	1	0	0



Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Each patient has a vector v where $v_i = 1$ if patient has a rare variant in gene i

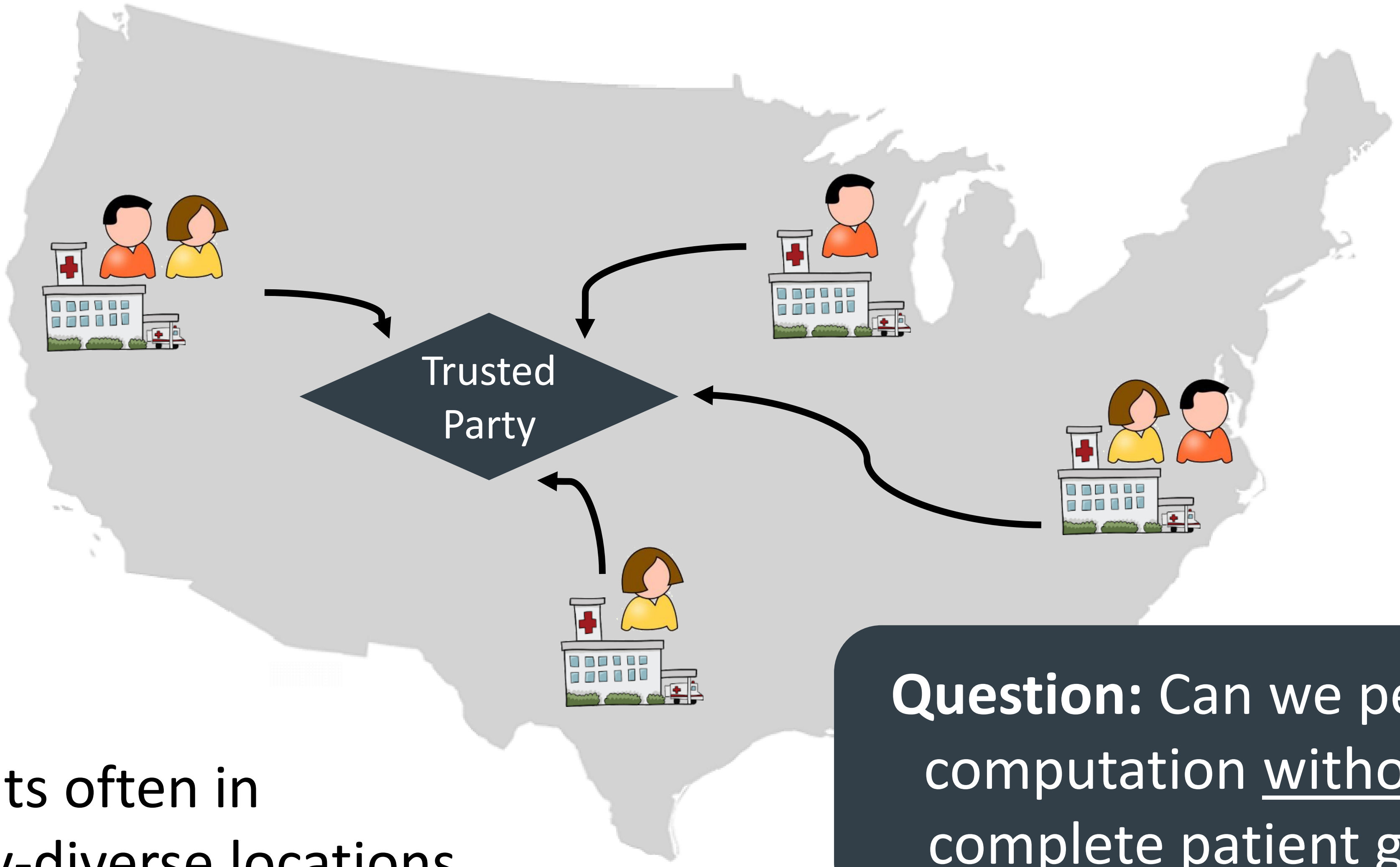
To identify causal rare variants, often need exact computation

Goal: Identify gene with most variants across all patients

Works well for Mendelian (monogenic) diseases (estimated to affect ≈10% of individuals)

Rare Disease Diagnosis

[JWBBB17]

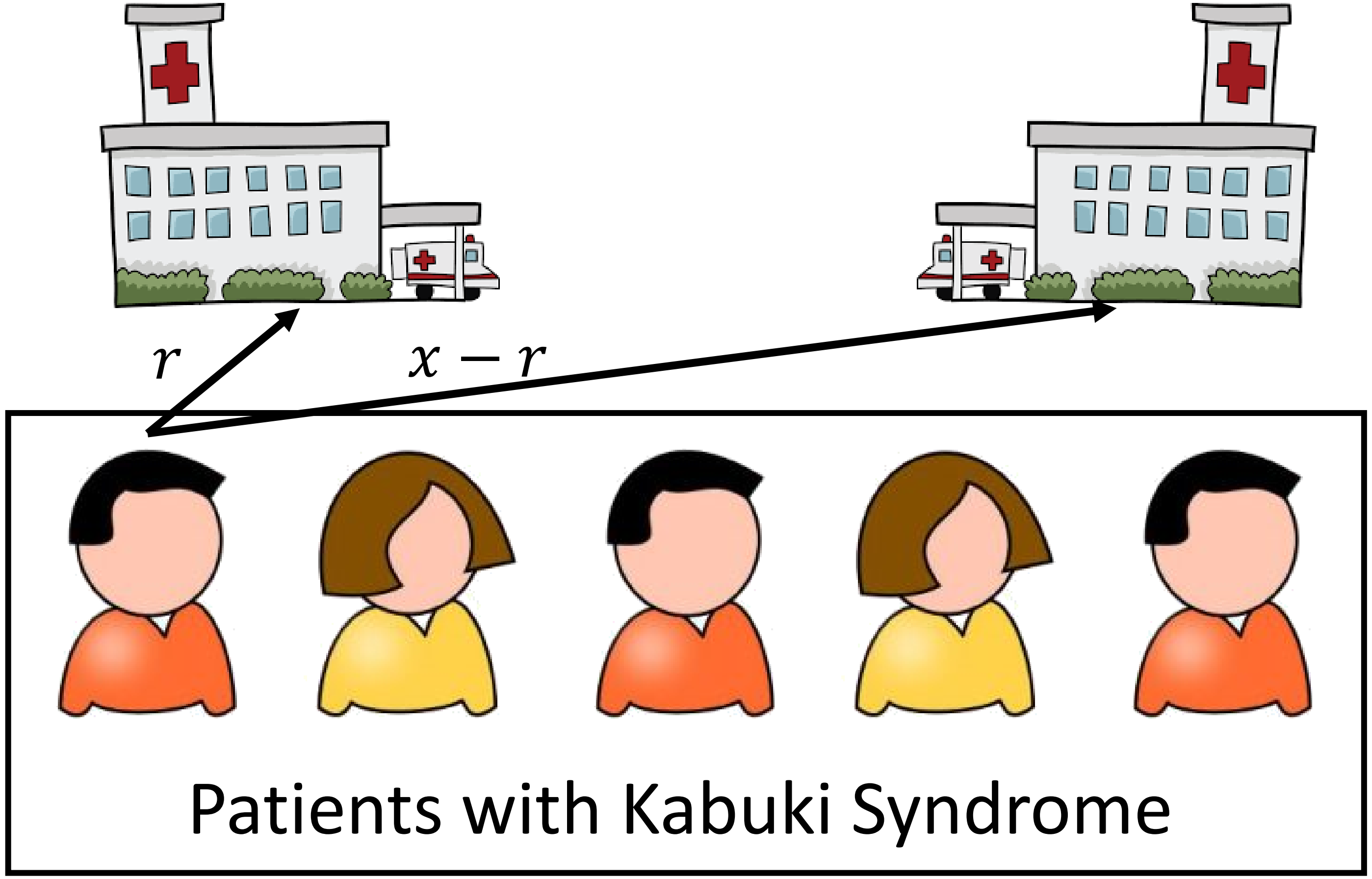


Patients often in geographically-diverse locations

Question: Can we perform this computation <u>without</u> seeing complete patient genomes?

Rare Disease Diagnosis

[JWBBB17]

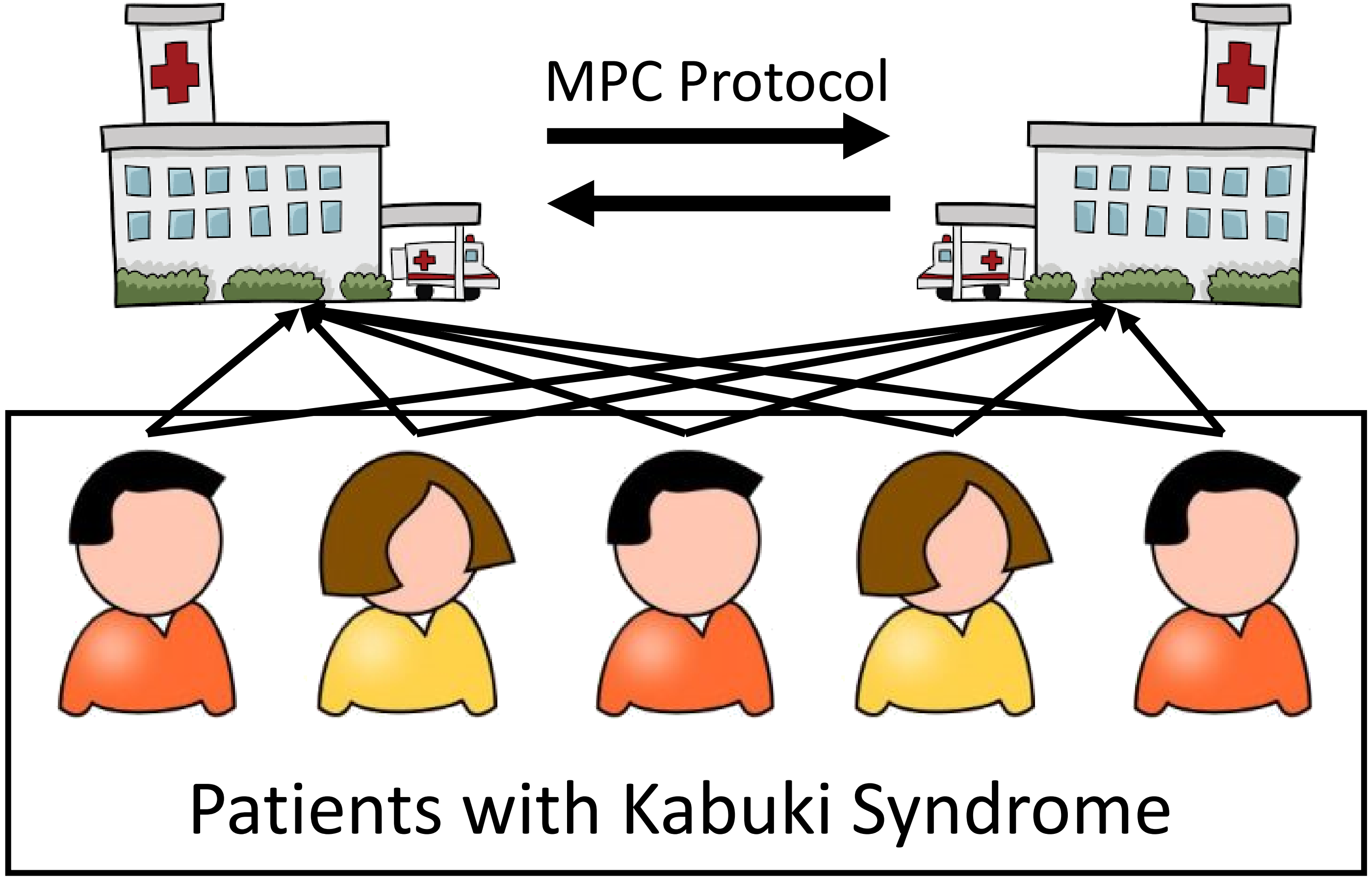


Patients "secret share" their data with two non-colluding hospitals

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Rare Disease Diagnosis

[JWBBB17]



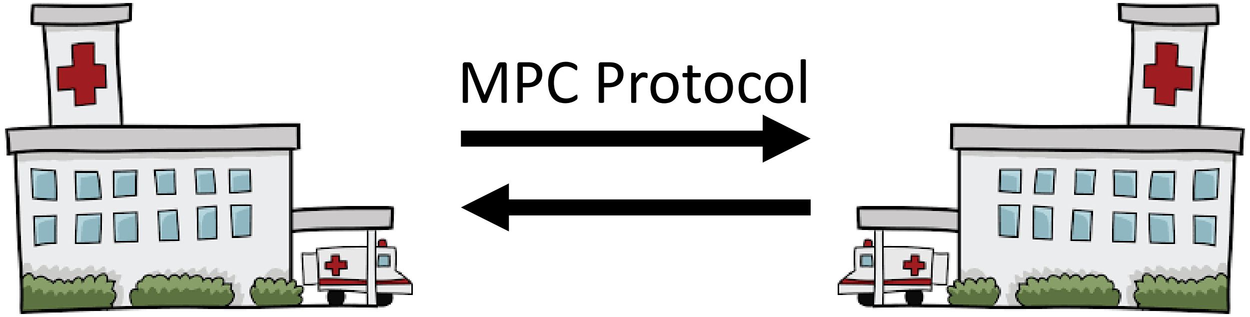
Hospitals run a multiparty computation (MPC) protocol on pooled inputs

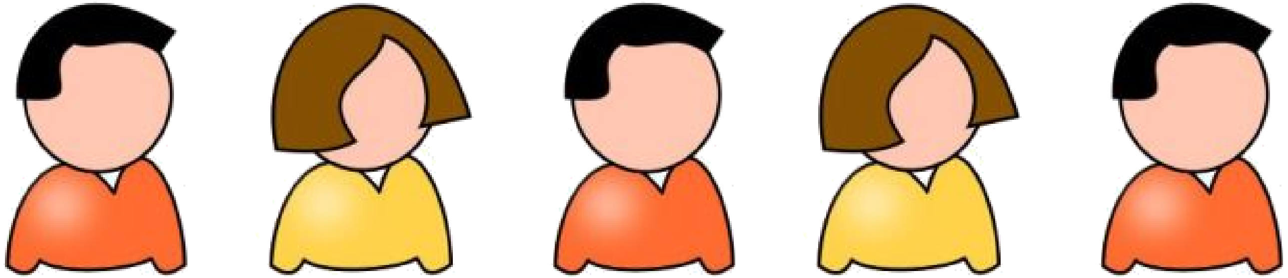
Patients "secret share" their data with two non-colluding hospitals

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Rare Disease Diagnosis

[JWBBB17]





Patients with Kabuki Syndrome

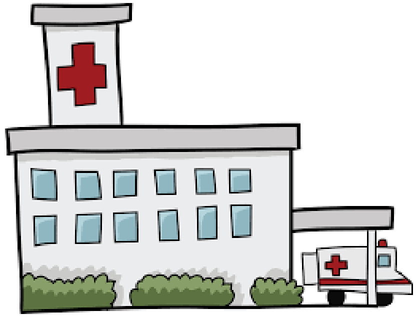
Each patient has a list of 200-400 rare variants over ≈20,000 genes

Top variants (sorted):
KMT2D, COL6A1, FLNB

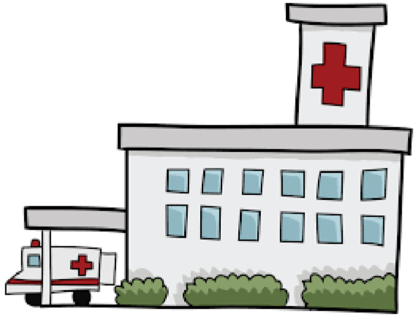
Known cause of disease

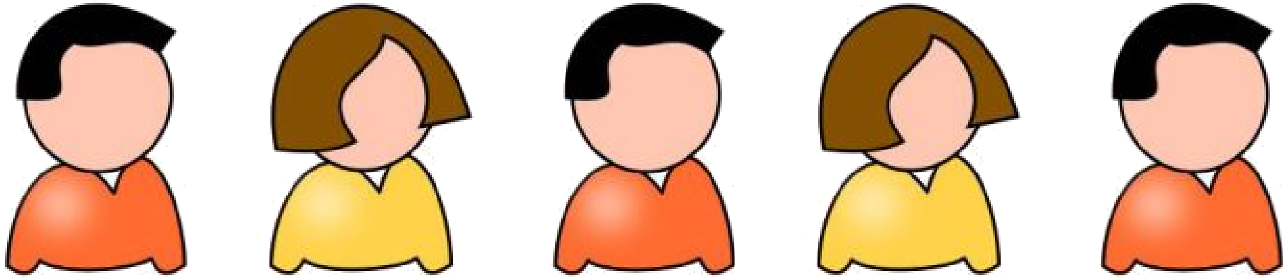
Rare Disease Diagnosis

[JWBBB17]



MPC Protocol





Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes

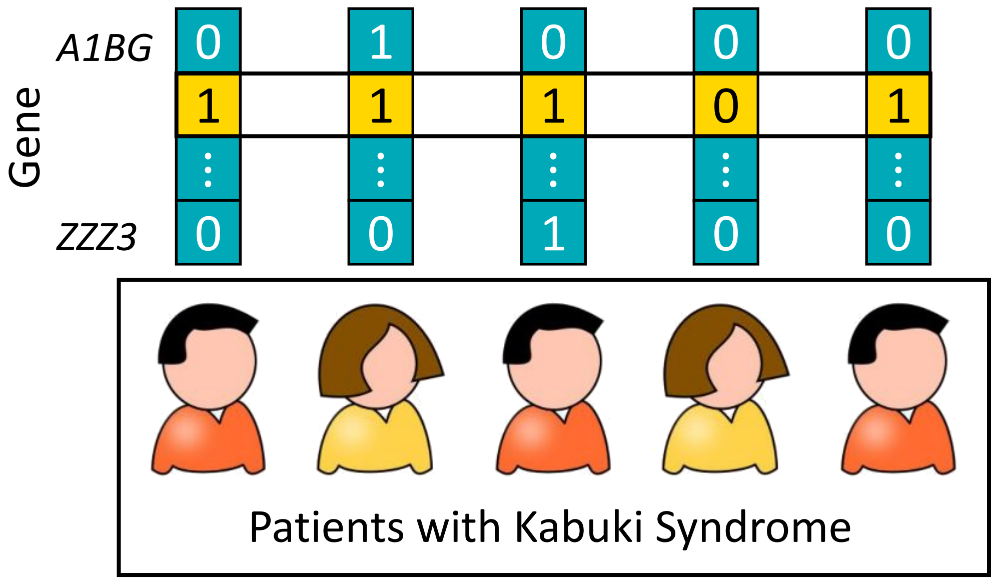
Top variants (sorted):
KMT2D, COL6A1, FLNB

Other variants that the patients possess are kept hidden

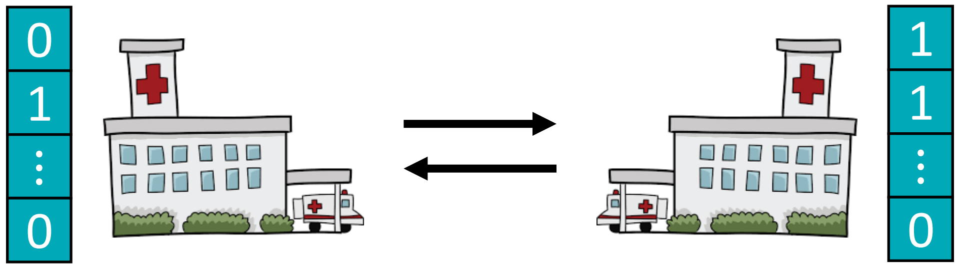
Rare Disease Diagnosis

[JWBBB17]

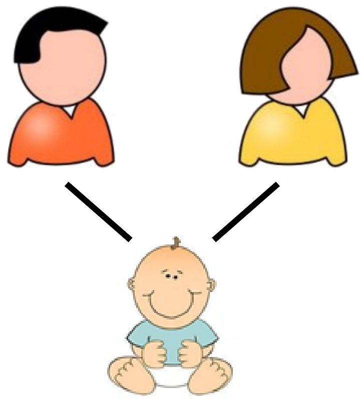
General techniques apply to many different scenarios for diagnosing Mendelian diseases



Identify causal gene for a rare disease given a small patient cohort



Identify patients with the same rare functional mutation at two different hospitals



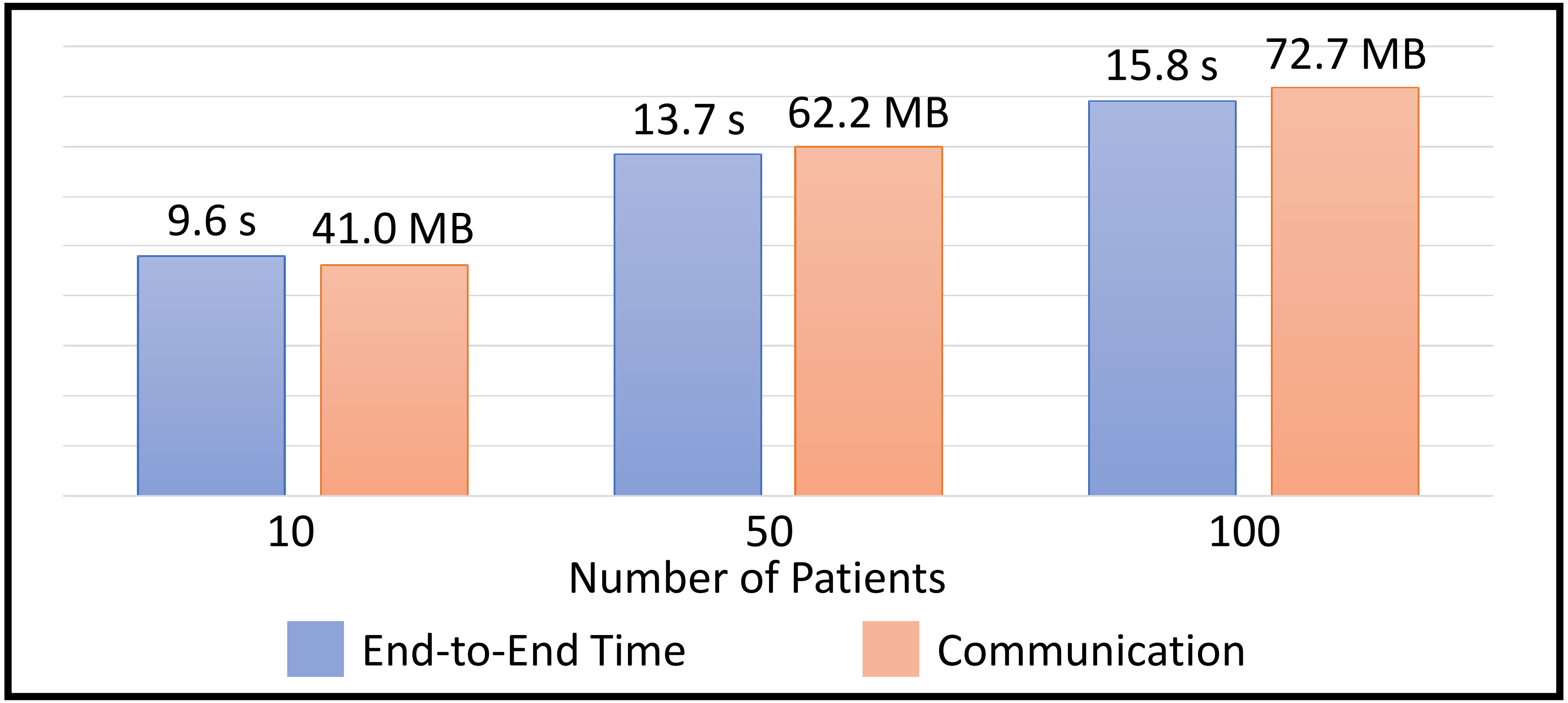
Identify rare functional variants that are present in the child but in neither of the parents

Rare Disease Diagnosis

[JWBBB17]

Experimental benchmarks for identifying causal gene in small disease cohort

- Simulated two non-colluding entities with 1 server on East Coast and 1 on West Coast

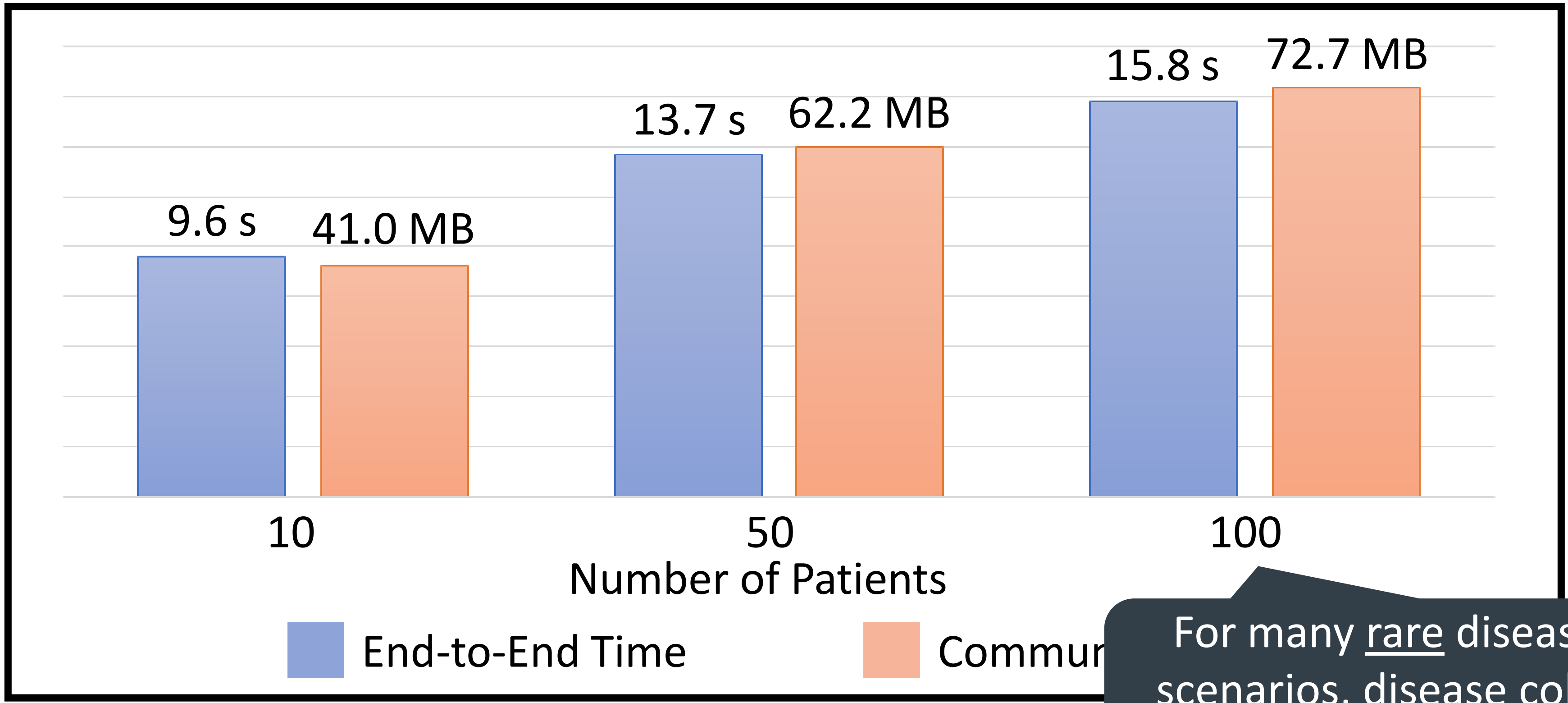


Rare Disease Diagnosis

[JWBBB17]

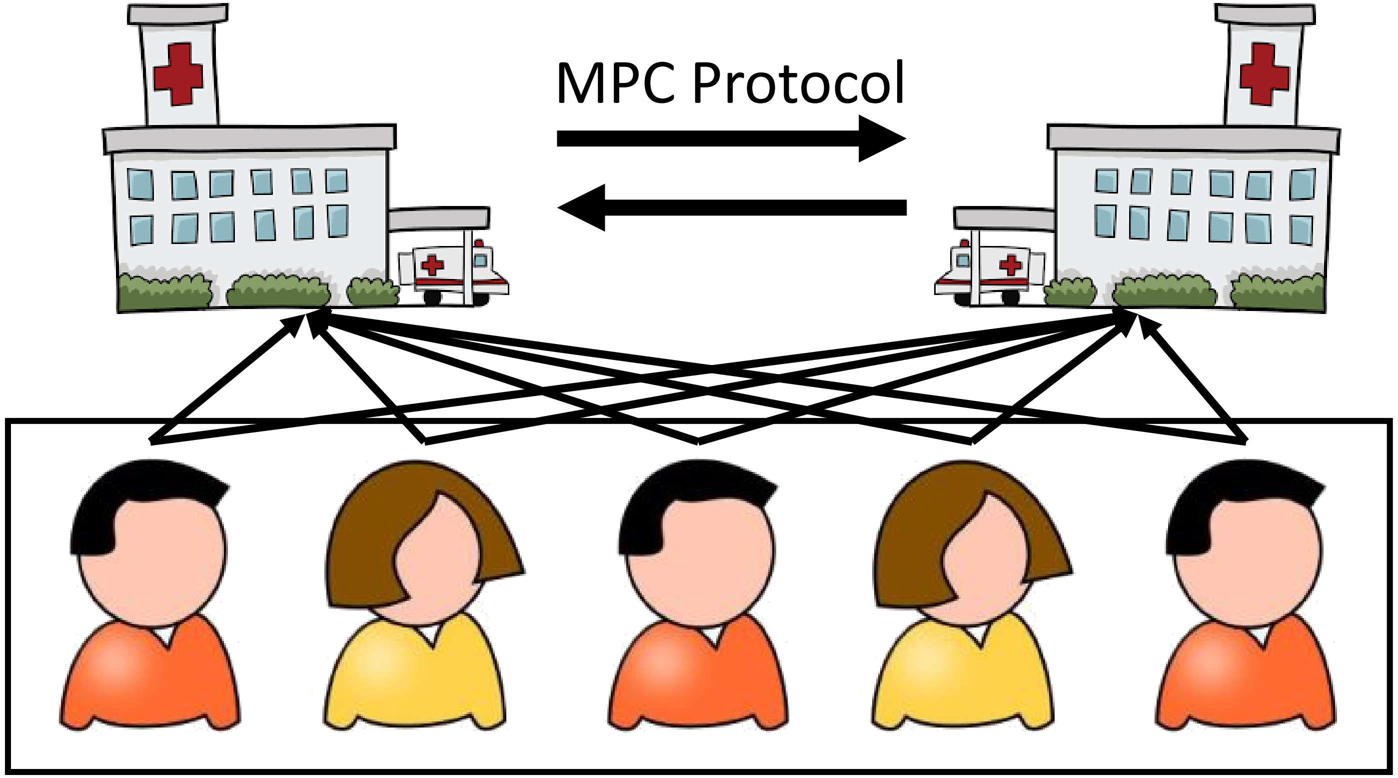
Experimental benchmarks for identifying causal gene in small disease cohort

- Simulated two non-colluding entities with 1 server on East Coast and 1 on West Coast



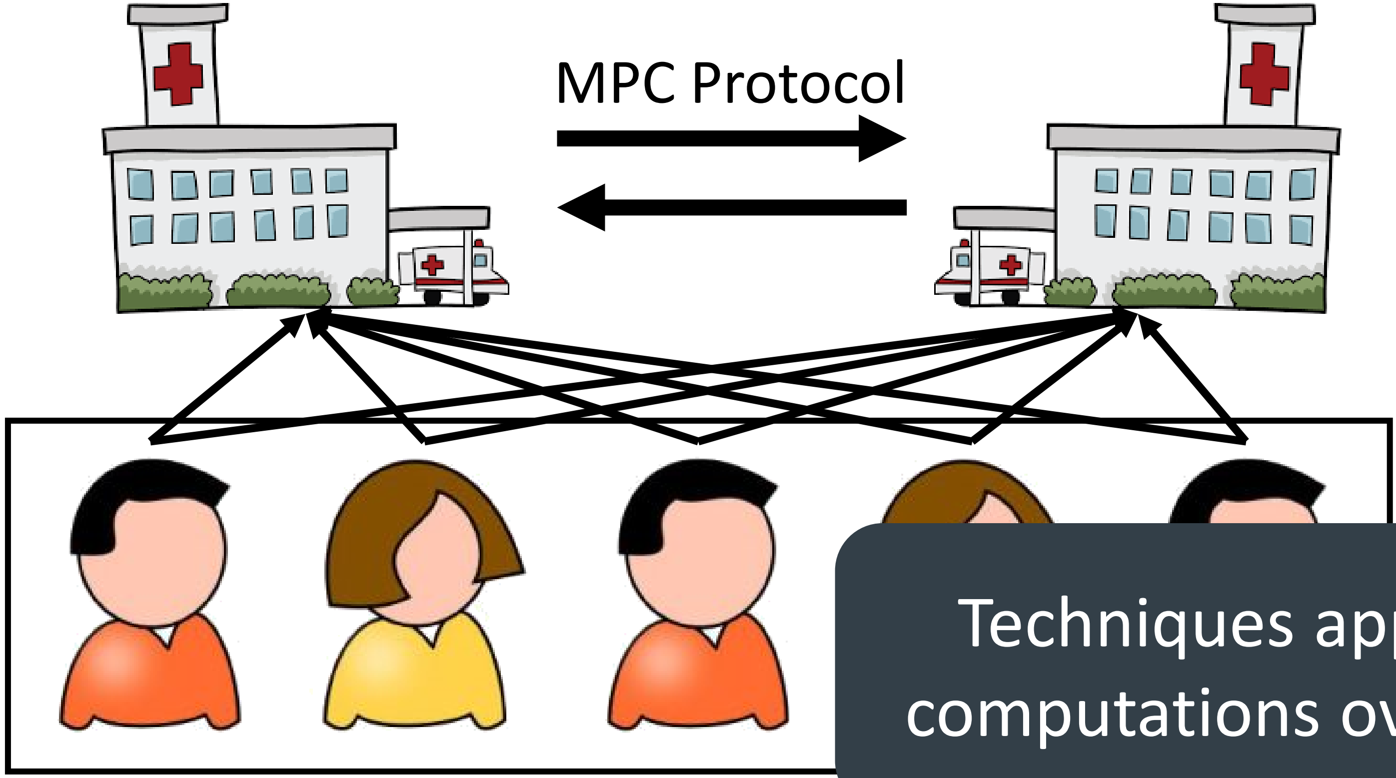
For many rare disease diagnosis scenarios, disease cohort size can be very small (e.g., 5-10 patients)

Secure Genome Computation



Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes

Secure Genome Computation



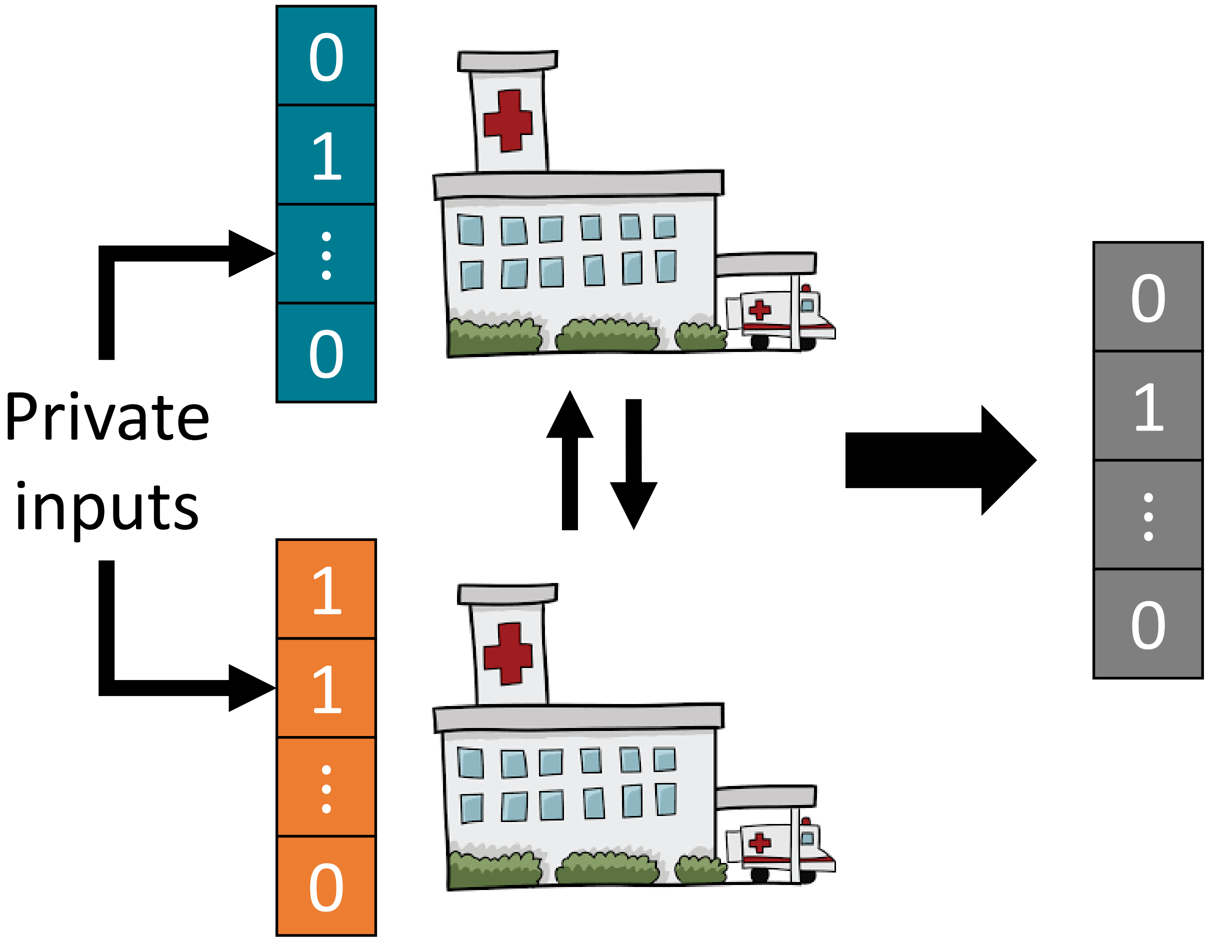
Techniques apply to <u>general</u> computations over <u>private</u> data

Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes

Yao's Protocol for Two-Party Computation

Yao's Protocol for Two-Party Computation

[Yao82]



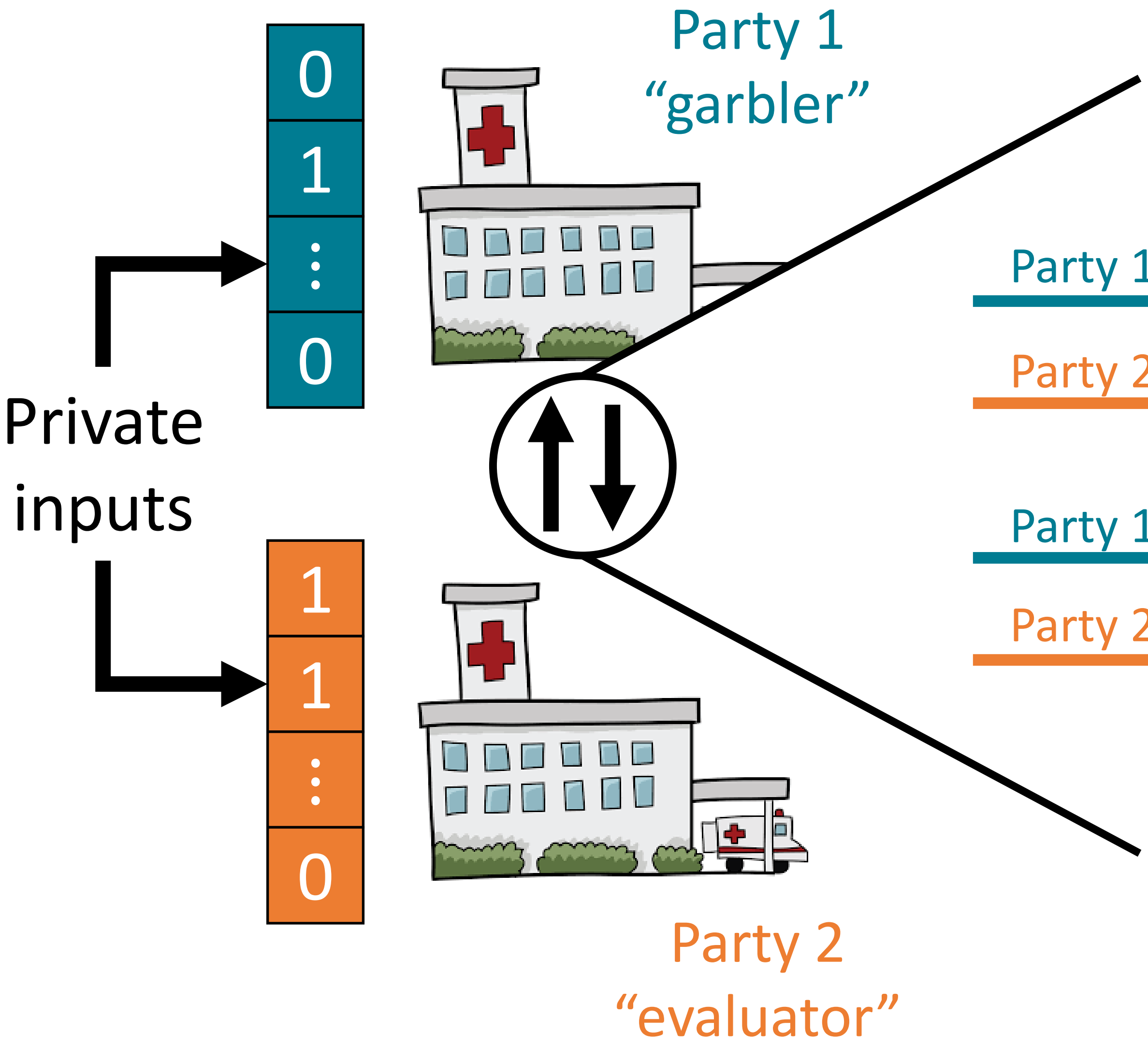
Security guarantee: everything the parties learn can be inferred from the output and their individual inputs

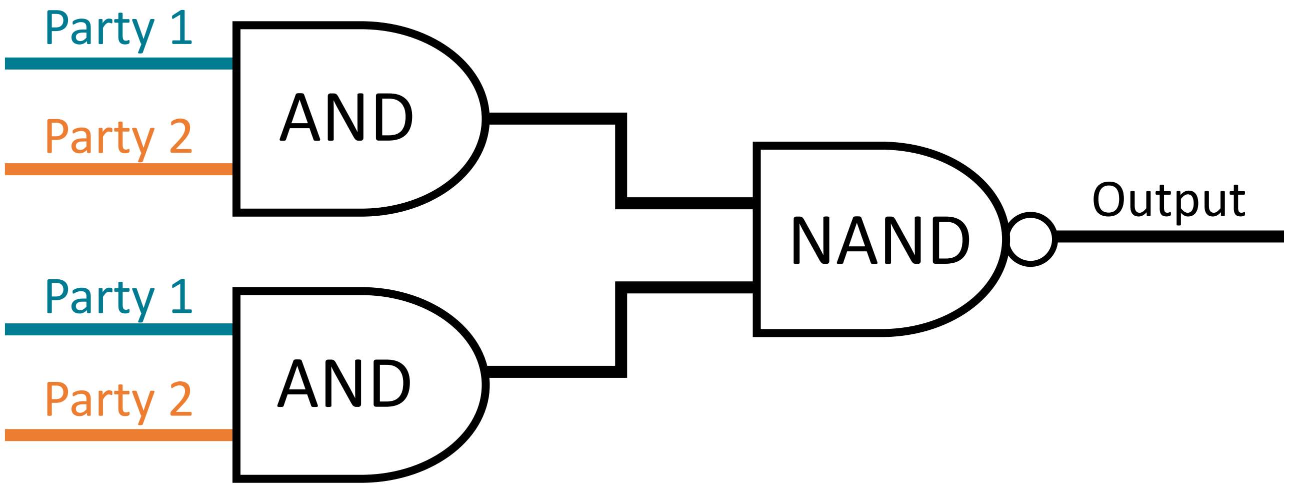
Classic protocol for two-party computation

Yao’s Protocol for Two-Party Computation

[Yao82]

Step 1: Model computation as a Boolean circuit

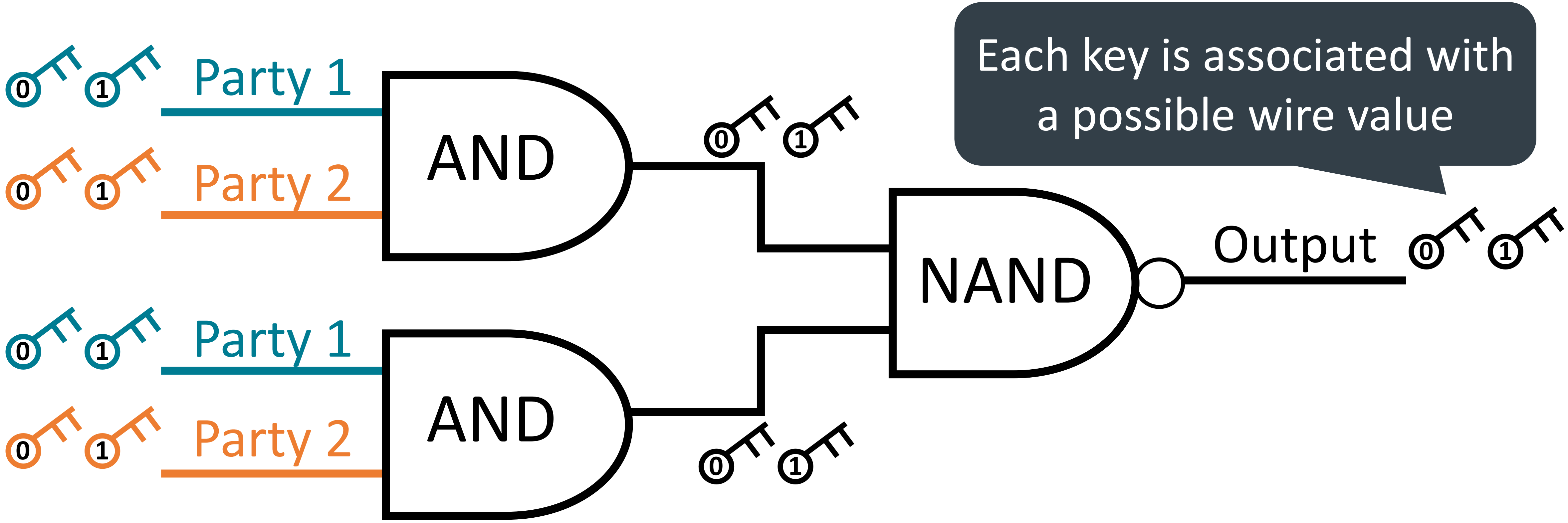




Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)

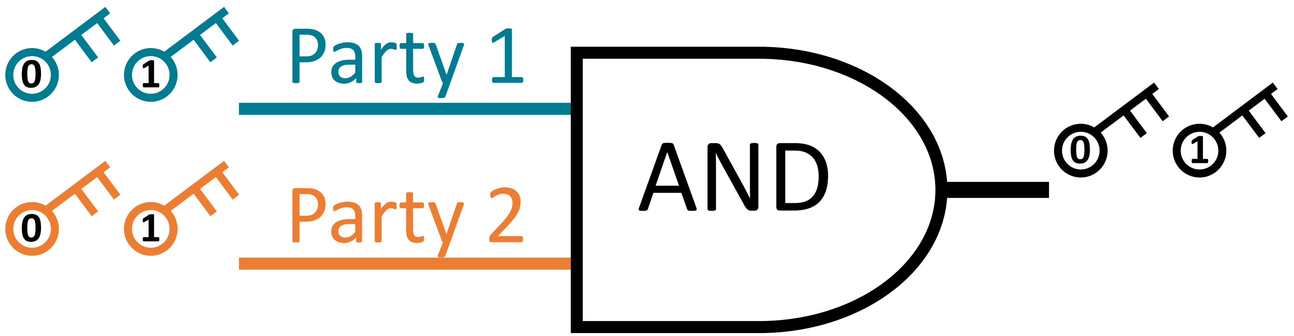


Garbler chooses two different encryption keys for every wire in the circuit

Yao’s Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)



Inputs		Output
Party 1	Party 2	
0	0	0
0	1	0
1	0	0
1	1	1

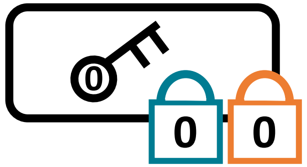
Idea: Encrypt the output key (for the output wire) with the two input keys (for the input wires)

Garbler constructs a garbled truth table for each gate in the circuit

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

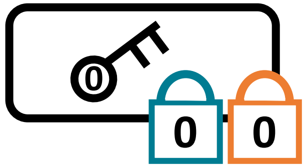
Inputs: Party 1	Inputs: Party 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Garbler constructs a garbled truth table for each gate in the circuit

Yao’s Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

Key for party 1’s input

Key for party 2’s input

Key for output wire

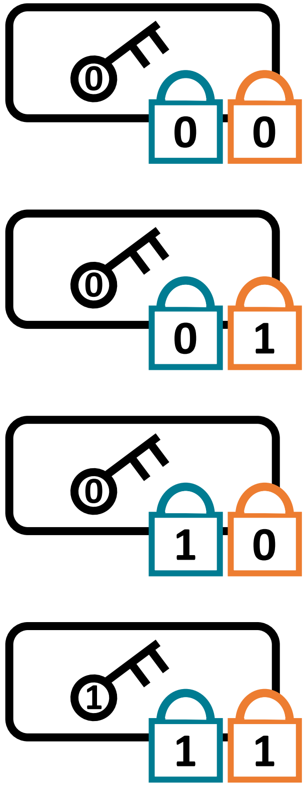
Inputs		Output
Party 1	Party 2	
0	0	0
0	1	0
1	0	0
1	1	1

Garbler constructs a garbled truth table for each gate in the circuit

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_1^{(\mathrm{out})}\right)\right)$$

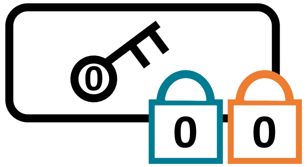
Inputs: Party 1	Inputs: Party 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Garbler constructs a garbled truth table for each gate in the circuit

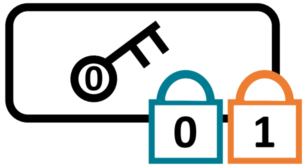
Yao's Protocol for Two-Party Computation

[Yao82]

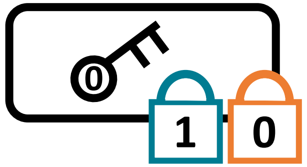
Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



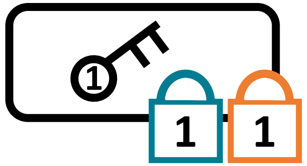
$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_1^{(\mathrm{out})}\right)\right)$$

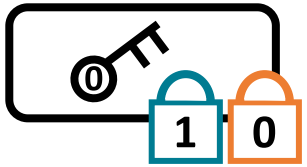
Garbled truth table randomly permuted

Garbler constructs a garbled truth table for each gate in the circuit

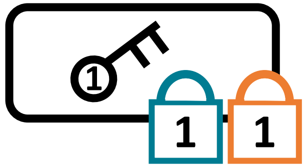
Yao’s Protocol for Two-Party Computation

[Yao82]

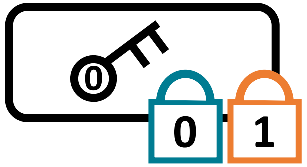
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)



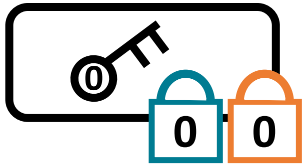
$$\text{Enc}\left(k_1^{(1)}, \text{Enc}\left(k_0^{(2)}, k_0^{(\text{out})}\right)\right)$$



$$\text{Enc}\left(k_1^{(1)}, \text{Enc}\left(k_1^{(2)}, k_1^{(\text{out})}\right)\right)$$



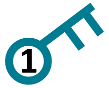
$$\text{Enc}\left(k_0^{(1)}, \text{Enc}\left(k_1^{(2)}, k_0^{(\text{out})}\right)\right)$$



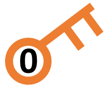
$$\text{Enc}\left(k_0^{(1)}, \text{Enc}\left(k_0^{(2)}, k_0^{(\text{out})}\right)\right)$$

Garbled truth table randomly permuted

Invariant: Given just a single key for each input wire, evaluator can learn a <u>single</u> key for the output wire



$k_1^{(1)}$



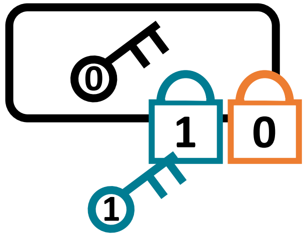
$k_0^{(2)}$

Garbler constructs a garbled truth table for each gate in the circuit

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)

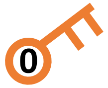


$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$

Garbled truth table randomly permuted

Invariant: Given just a single key for each input wire, evaluator can learn a <u>single</u> key for the output wire

$k_1^{(1)}$

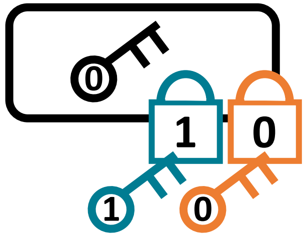


$k_0^{(2)}$

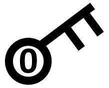
Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$



$k_0^{(\mathrm{out})}$

$k_0^{(\mathrm{out})}$ is just a symmetric key – does not reveal what the output bit is

Garbled truth table randomly permuted

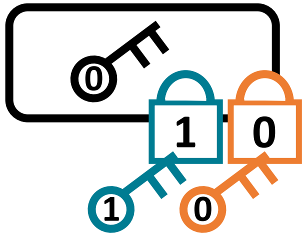
Invariant: Given just a single key for each input wire, evaluator can learn a single key for the output wire

$k_1^{(1)}$ $k_0^{(2)}$

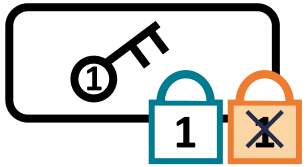
Yao's Protocol for Two-Party Computation

[Yao82]

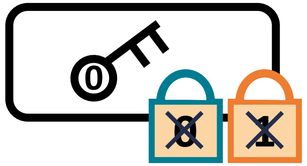
Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



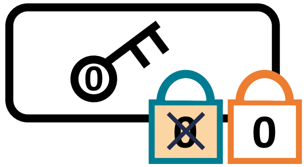
$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_1^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_1^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_1^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$



$$\mathrm{Enc}\left(k_0^{(1)}, \mathrm{Enc}\left(k_0^{(2)}, k_0^{(\mathrm{out})}\right)\right)$$

Cannot decrypt other output keys

Garbled truth table randomly permuted

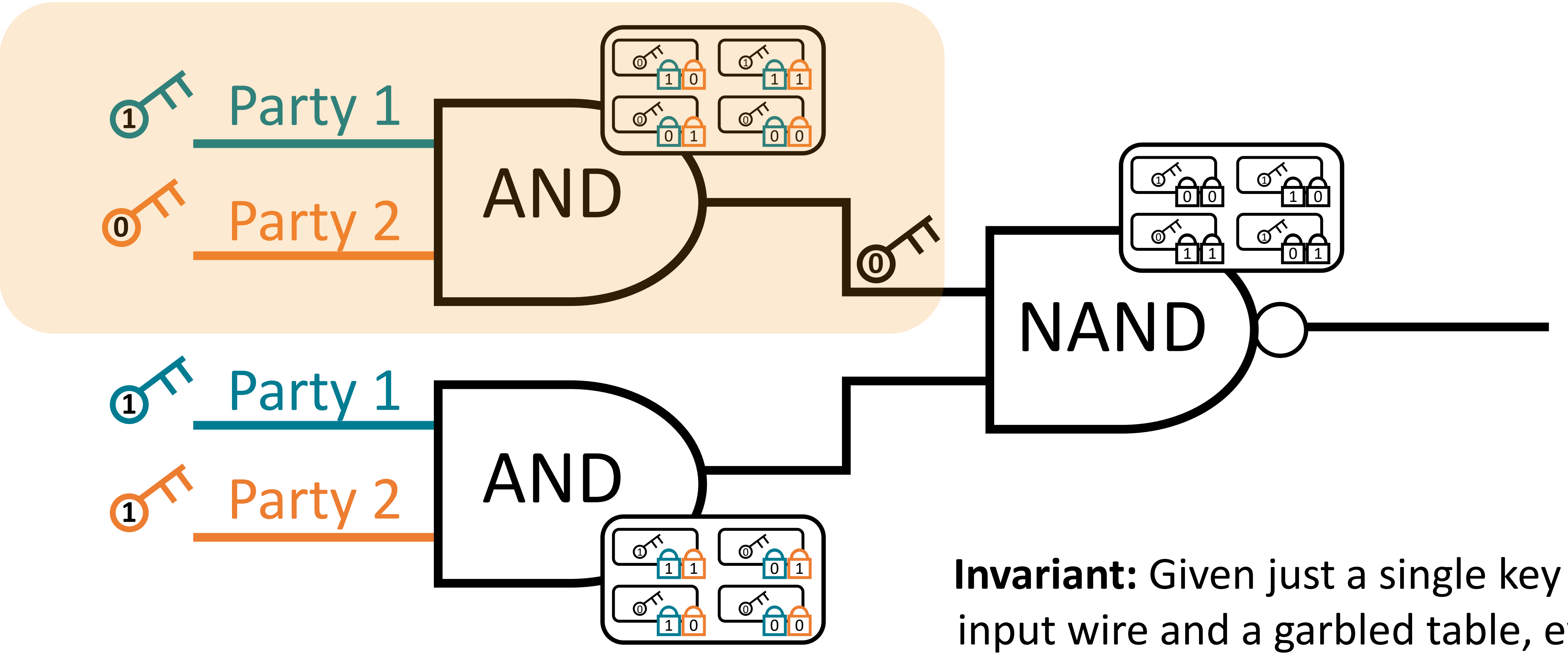
Invariant: Given just a single key for each input wire, evaluator can learn a <u>single</u> key for the output wire

$k_1^{(1)}$ $k_0^{(2)}$

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)

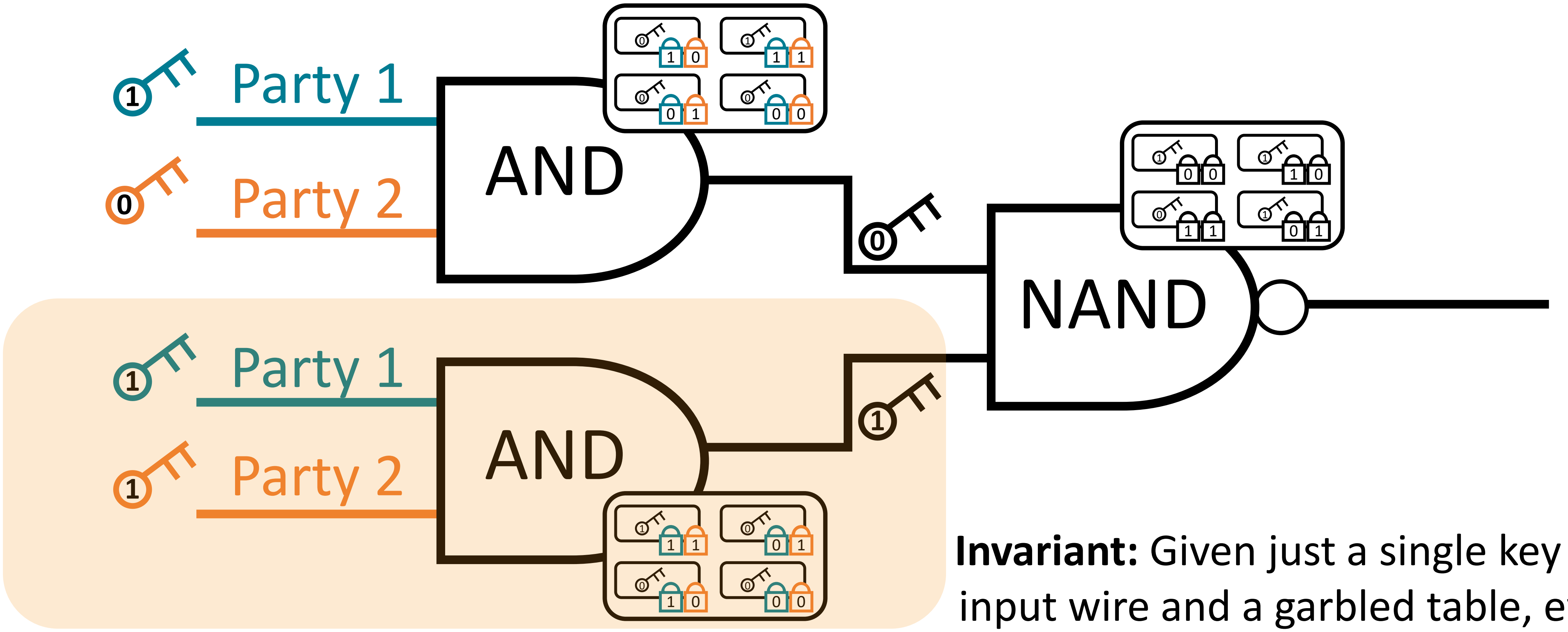


Invariant: Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)

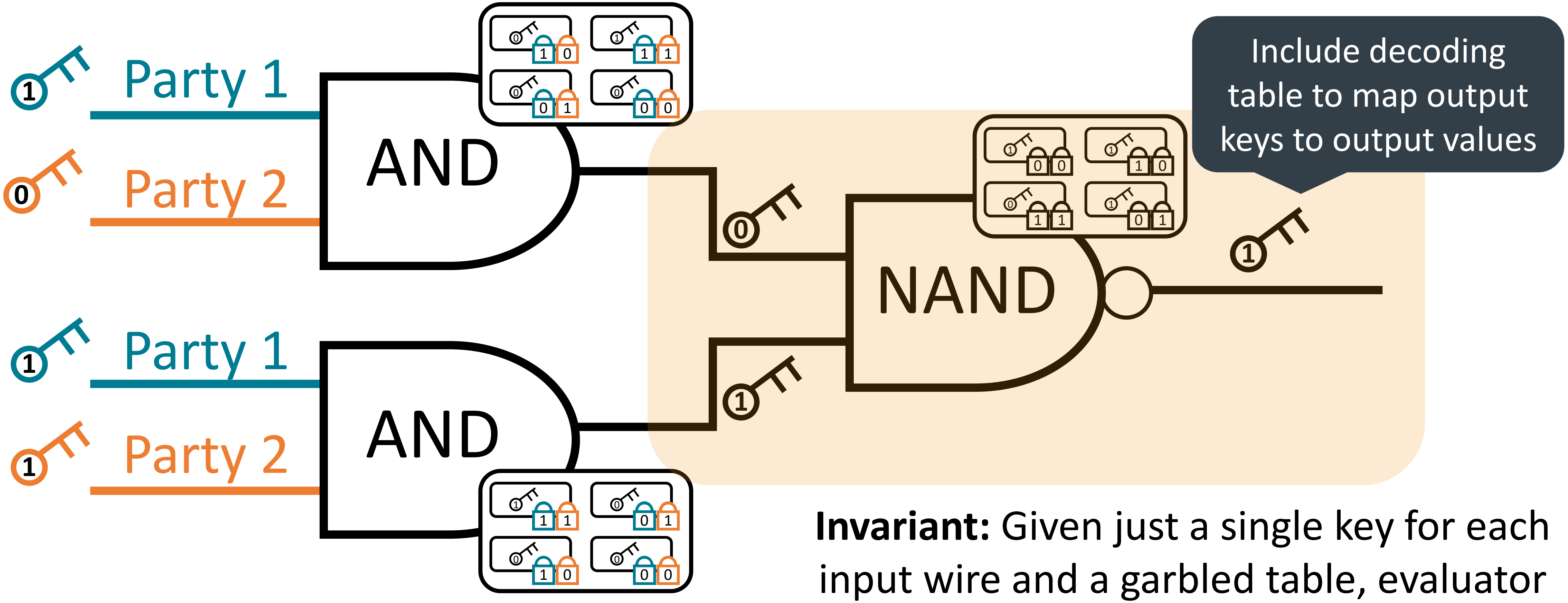


Invariant: Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)

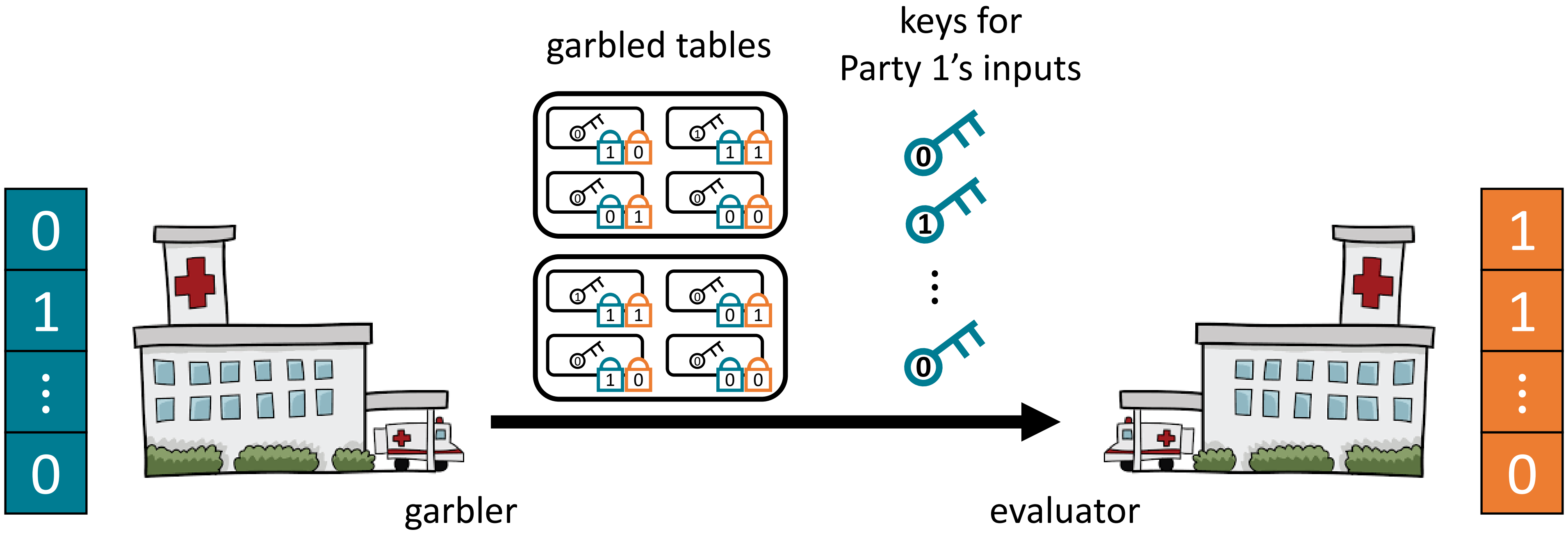


Invariant: Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire

Yao's Protocol for Two-Party Computation

[Yao82]

Step 2: Garbler "encrypts" the circuit (i.e., "garbles" the circuit)



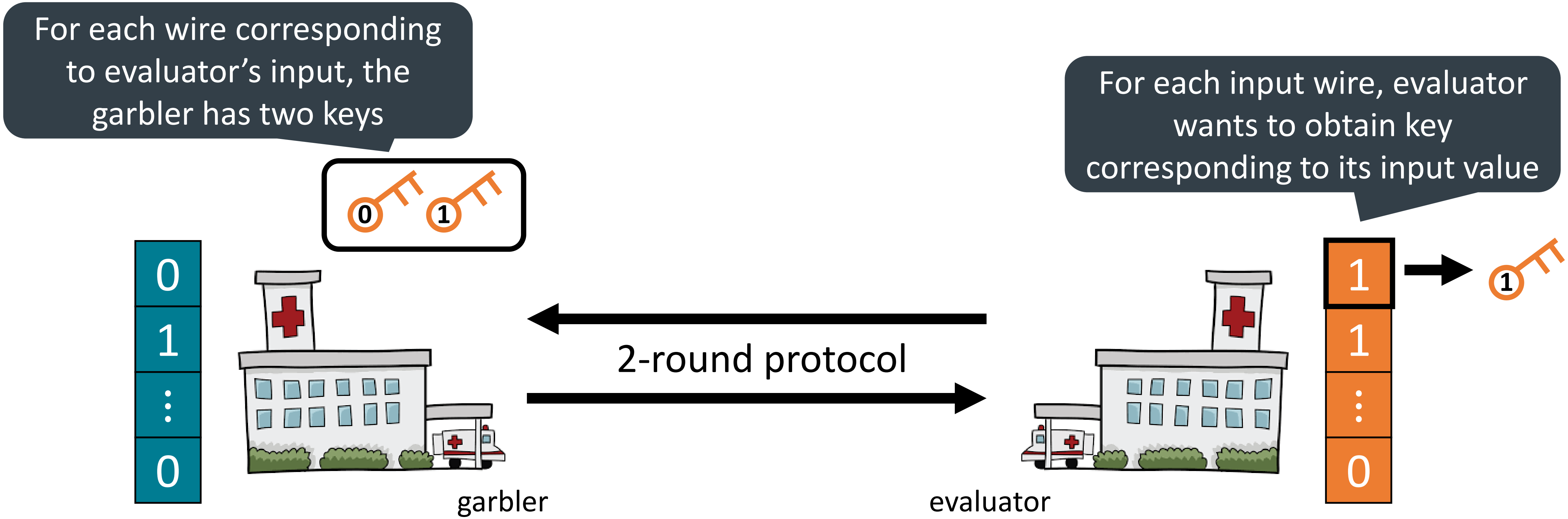
Question: how does evaluator obtain keys for its input?

Garbler can send garbled truth tables and keys for its inputs

Yao's Protocol for Two-Party Computation

[Yao82]

Step 3: Evaluator uses "oblivious transfer" to obtain keys for its input



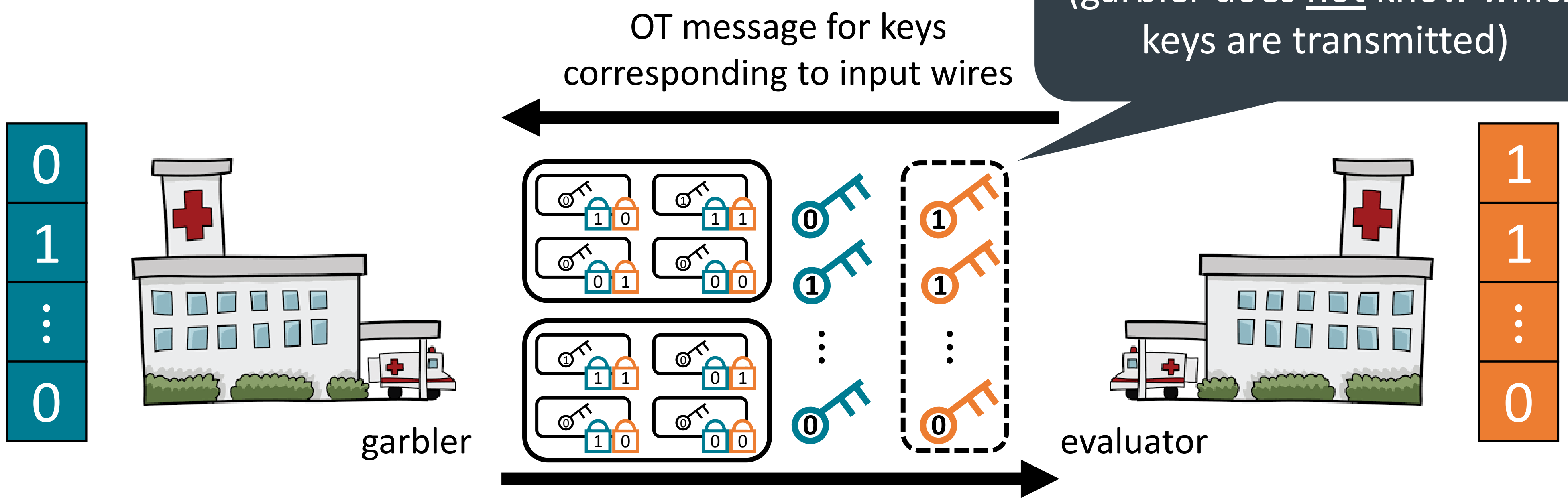
At the end of the oblivious transfer protocol, garbler learns nothing about which key evaluator obtains, and evaluator learns exactly one of the two keys

Yao’s Protocol for Two-Party Computation

[Yao82]

Two-round protocol for secure two-party communication

Keys communicated using OT (garbler does not know which keys are transmitted)



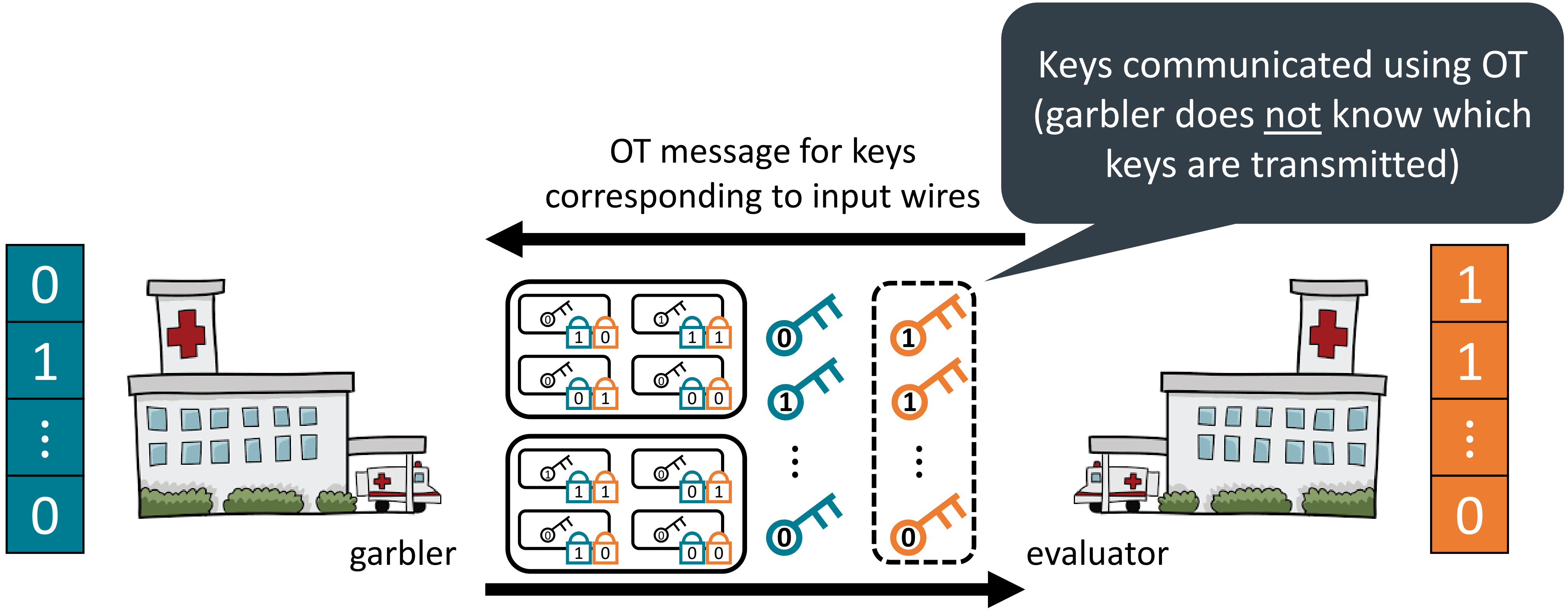
Many improvements are possible to achieve better performance

Evaluator uses keys to evaluate circuit gate-by-gate

Yao's Protocol for Two-Party Computation

[Yao82]

Two-round protocol for secure two-party communication



Many improvements are possible to achieve better performance

Protocol is very efficient; communication is the bottleneck

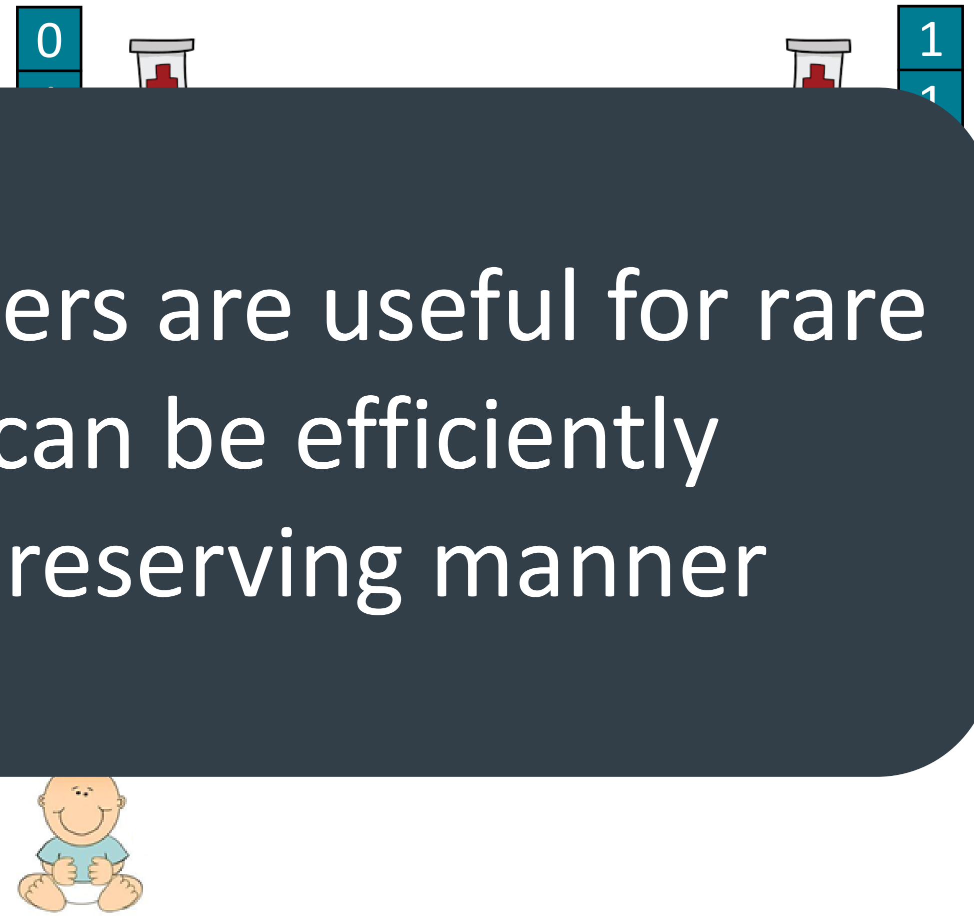
The Story So Far…

[JWBBB17]

General techniques apply to many different scenarios for diagnosing Mendelian diseases

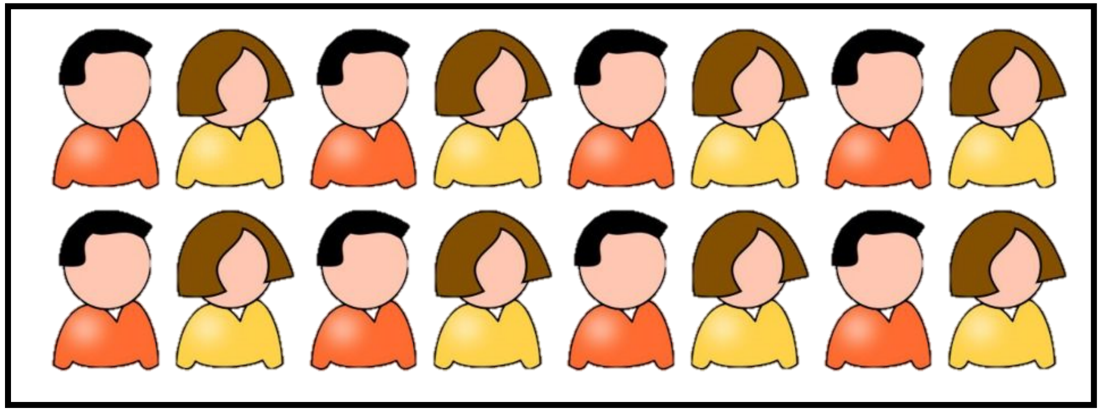
Simple frequency-based filters are useful for rare disease diagnosis and can be efficiently evaluated in a privacy-preserving manner

given a small patient cohort

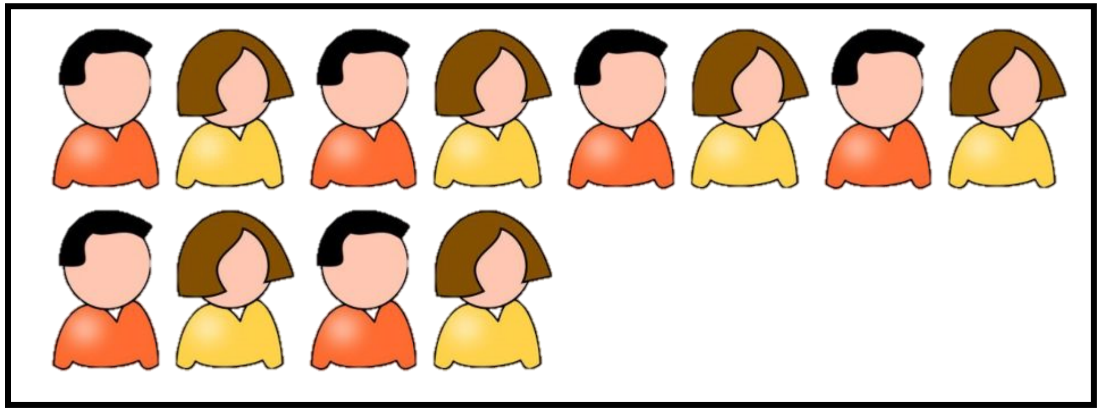


But What About More Complex Diseases?

[CWB18]



Control group (healthy)



Case group (affected)

Genome-wide association studies (GWAS):

- Identify genetic variants most correlated with a particular disease (or particular phenotype)
- Oftentimes, focused on identifying complex interactions between many variants

But What About More Complex Diseases?

[CWB18]

Each patient has a vector of SNPs (variations in specific locations in genome – 3 types)

Disease status

0	1	0	0	2	2	⋯	1		0
0	1	1	0	2	2	⋯	1		0
1	1	0	0	2	2	⋯	0		0
2	1	1	2	2	0	⋯	0		1
2	1	0	2	2	1	⋯	1		1

Goal: identify SNPs that are most correlated with disease status

Healthy individuals

Patients with lung cancer

But What About More Complex Diseases?

[CWB18]

Each patient has a vector of SNPs (variations in specific locations in genome – 3 types)

								Disease status
0	1	0	0	2	2	…	1	0
0	1	1	0	2	2	…	1	0
1	1	0	0	2	2	…	0	0
2	1	1	2	2	0	…	0	1
2	1	0	2	2	1	…	1	1

Goal: identify SNPs that are most correlated with disease status

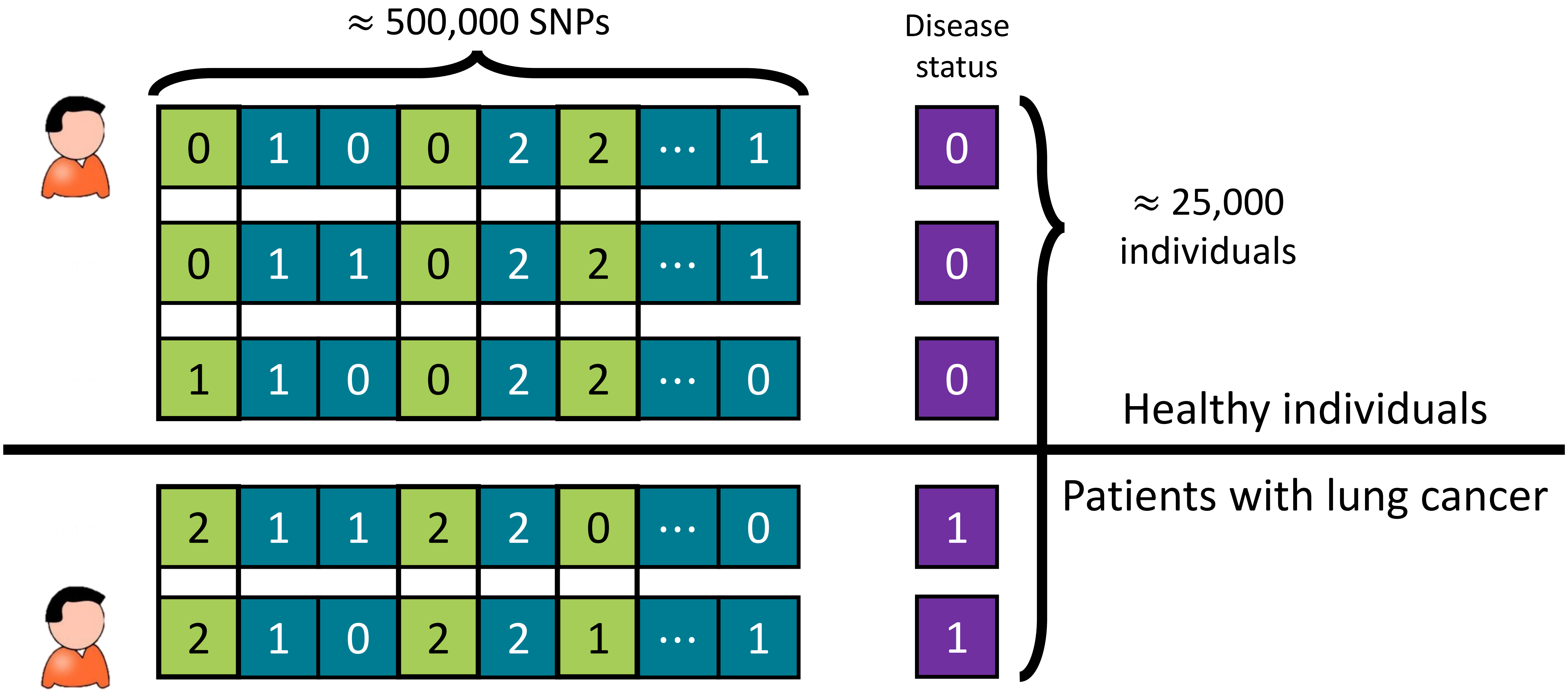
Healthy individuals

Patients with lung cancer

Unlike Mendelian diseases, we are looking for *many* associations (e.g., several hundred)

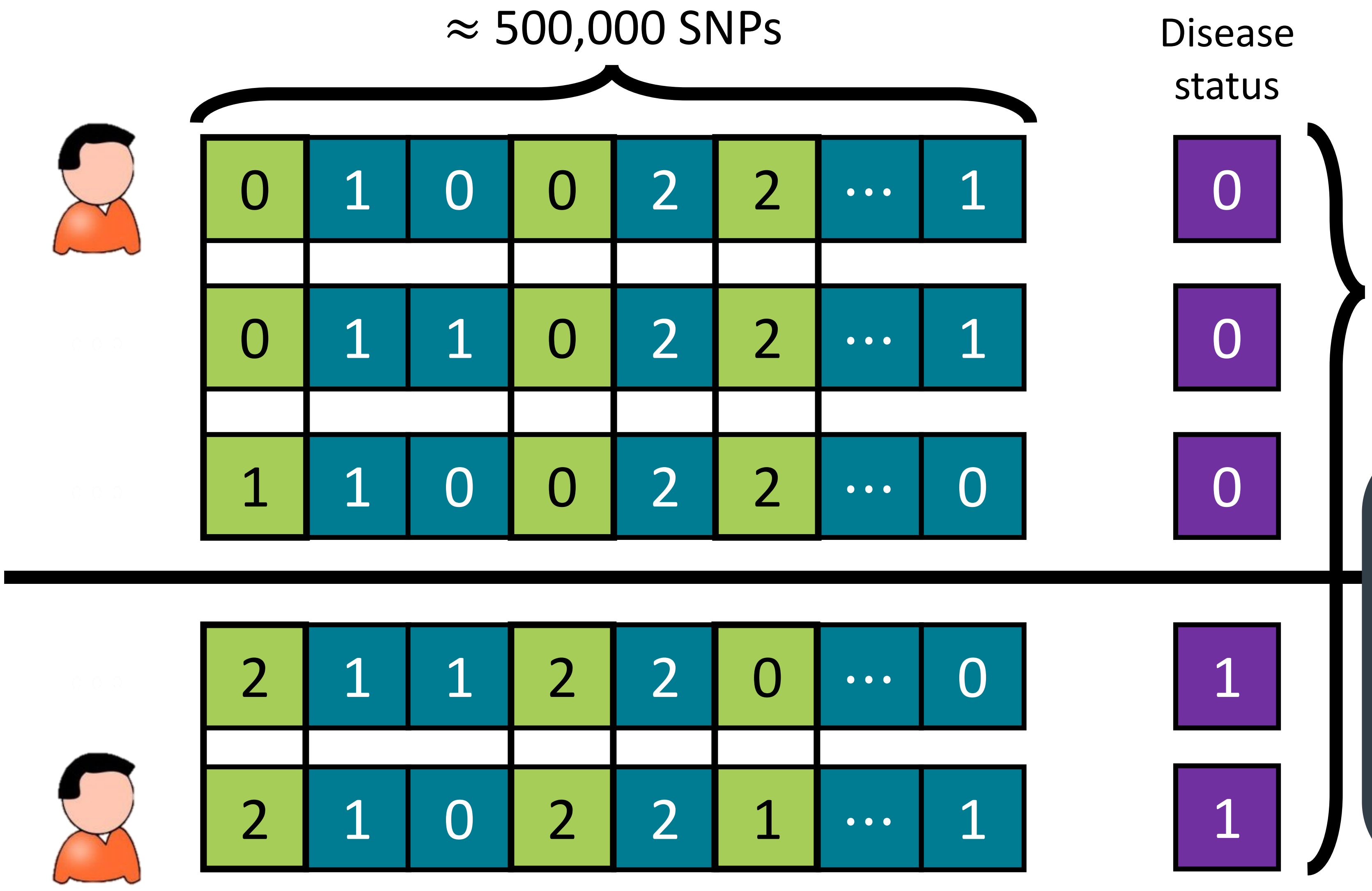
But What About More Complex Diseases?

[CWB18]



But What About More Complex Diseases?

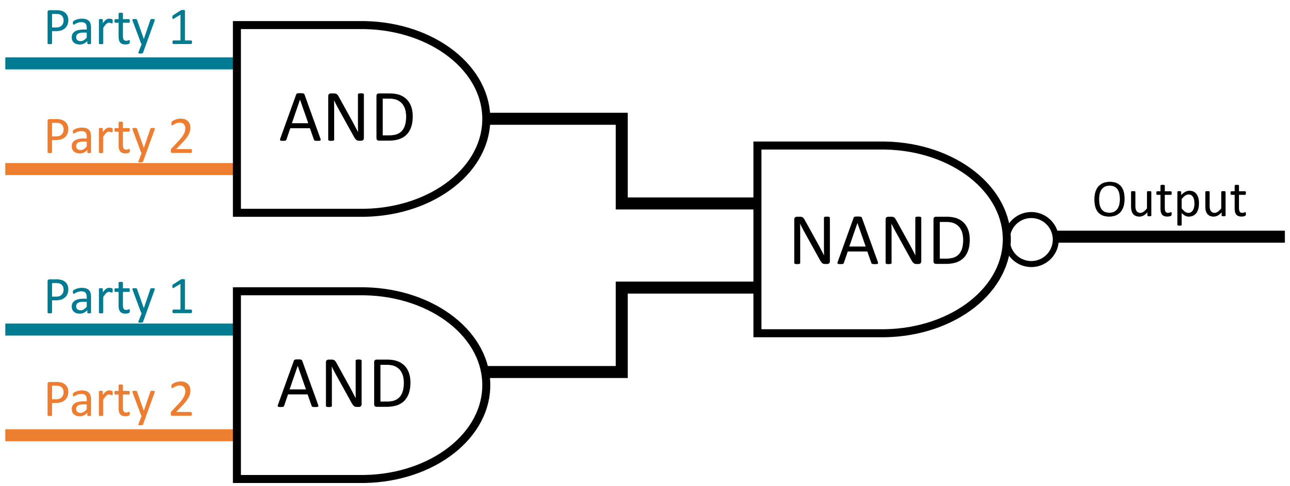
[CWB18]



Challenge: in real GWAS studies, we need to correct for *population-level* differences between groups

Arithmetic Computations on Shared Data

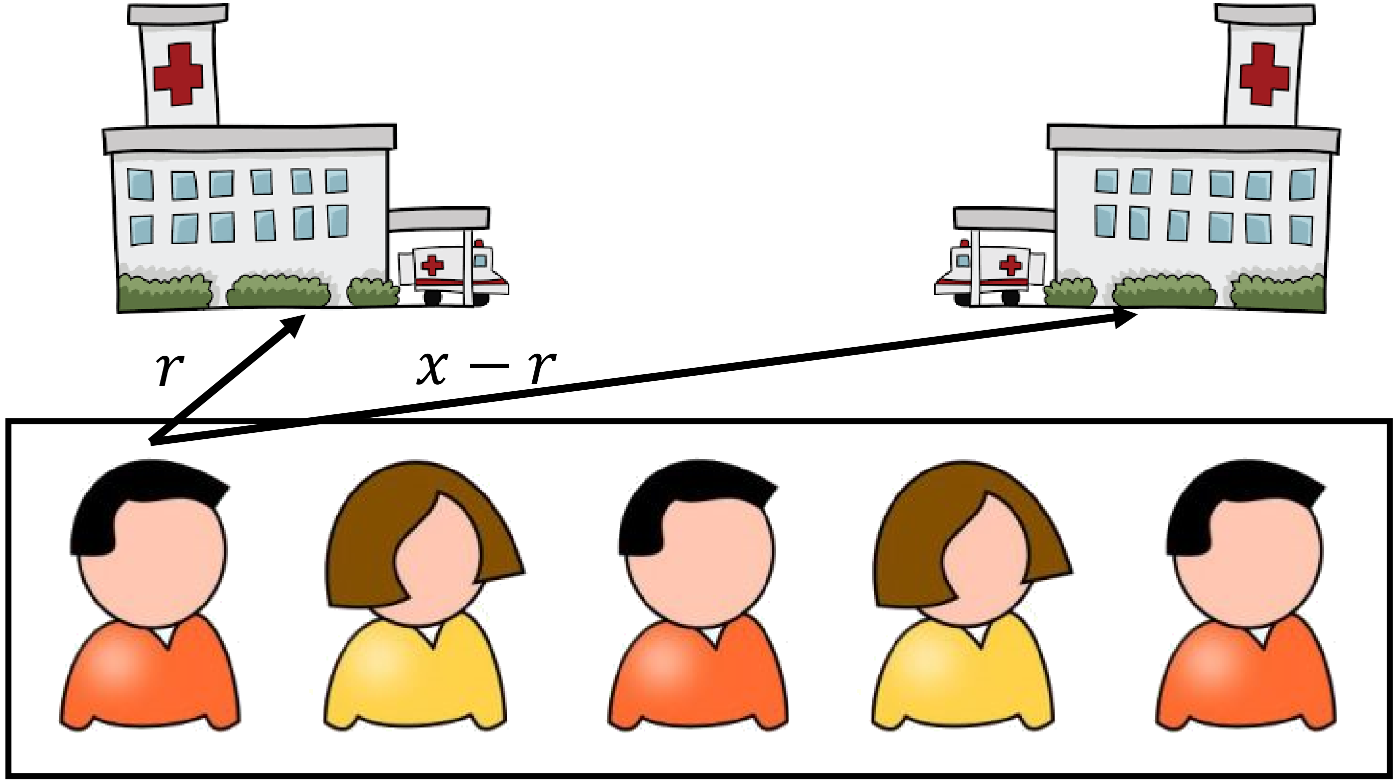
GWAS computations most naturally expressed as *arithmetic* computations (e.g., matrix operations)



Recall: to apply Yao's protocol, must first represent computation as a Boolean circuit

Can introduce significant overhead for arithmetic computations!

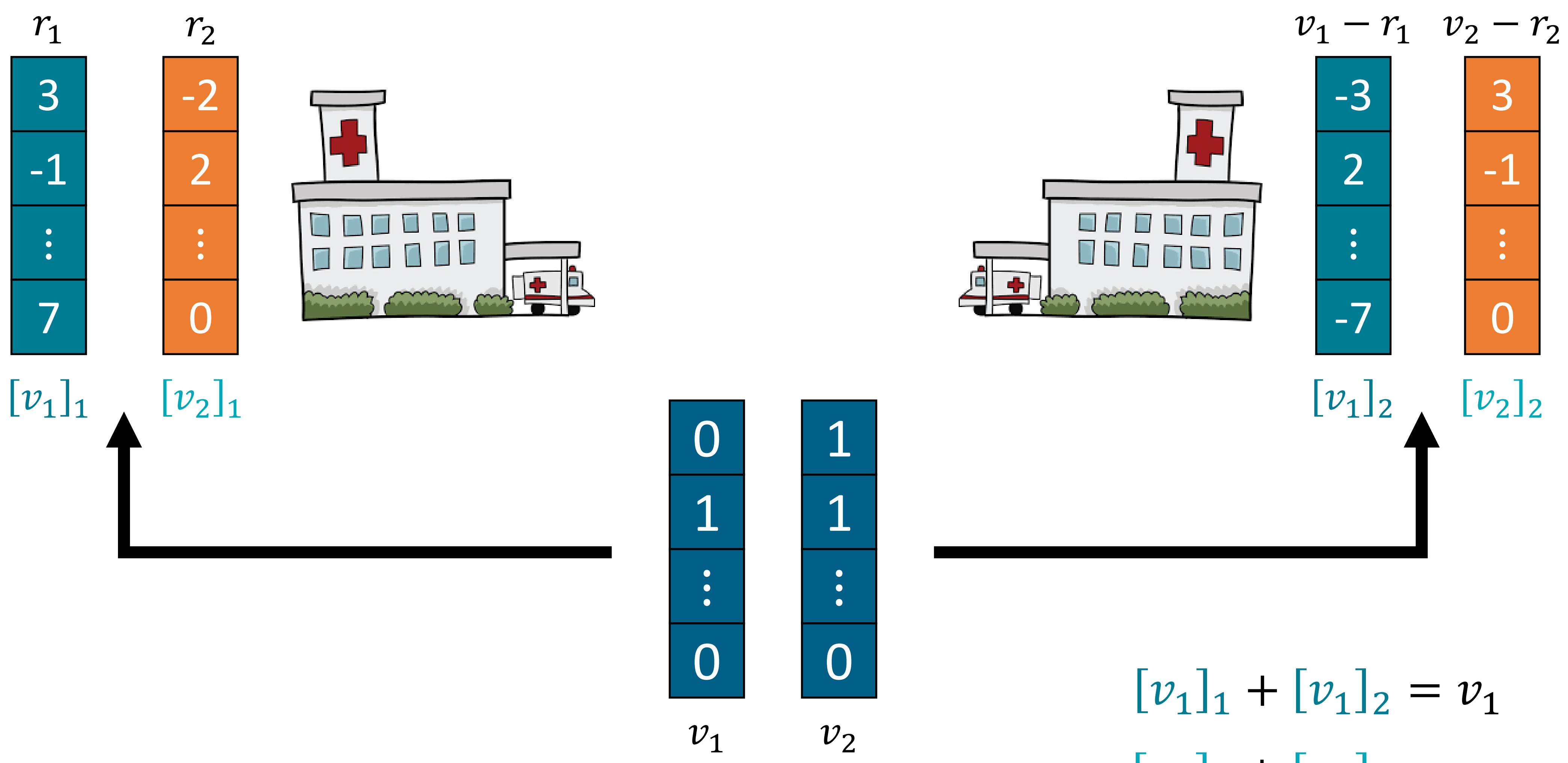
Arithmetic Computations on Shared Data



Patients “secret share” their data with two non-colluding hospitals

Approach: directly compute on secret-shared data

Arithmetic Computations on Shared Data



$$[v_1]_1 + [v_1]_2 = v_1$$

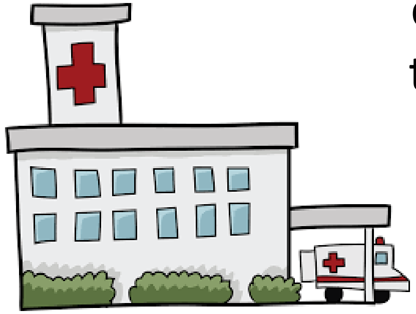
$$[v_2]_1 + [v_2]_2 = v_2$$

All operations done over a ring ($\mathbb{Z}_p$)

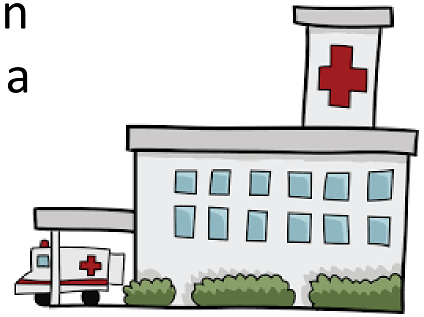
Arithmetic Computations on Shared Data

r_1: 3, -1, ⋮, 7 — $[v_1]_1$

r_2: -2, 2, ⋮, 0 — $[v_2]_1$



Observation: each party can locally compute on their shares to obtain a share of the sum



$v_1 - r_1$: -3, 2, ⋮, -7 — $[v_1]_2$

$v_2 - r_2$: 3, -1, ⋮, 0 — $[v_2]_2$

v_1: 0, 1, ⋮, 0

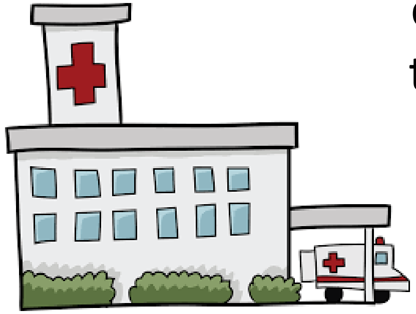
v_2: 1, 1, ⋮, 0

1, 1, ⋮, 7 — $[v_1]_1 + [v_2]_1 = [v_1 + v_2]_1$

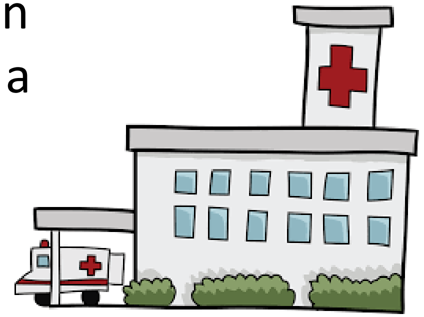
0, 1, ⋮, -7 — $[v_1]_2 + [v_2]_2 = [v_1 + v_2]_2$

Arithmetic Computations on Shared Data

r_1	r_2
3	-2
-1	2
⋮	⋮
7	0
$[v_1]_1$	$[v_2]_1$



Observation: each party can locally compute on their shares to obtain a share of the sum



$v_1 - r_1$	$v_2 - r_2$
-3	3
2	-1
⋮	⋮
-7	0
$[v_1]_2$	$[v_2]_2$

1
1
⋮
7

$[v_1]_1 + [v_2]_1 = [v_1 + v_2]_1$

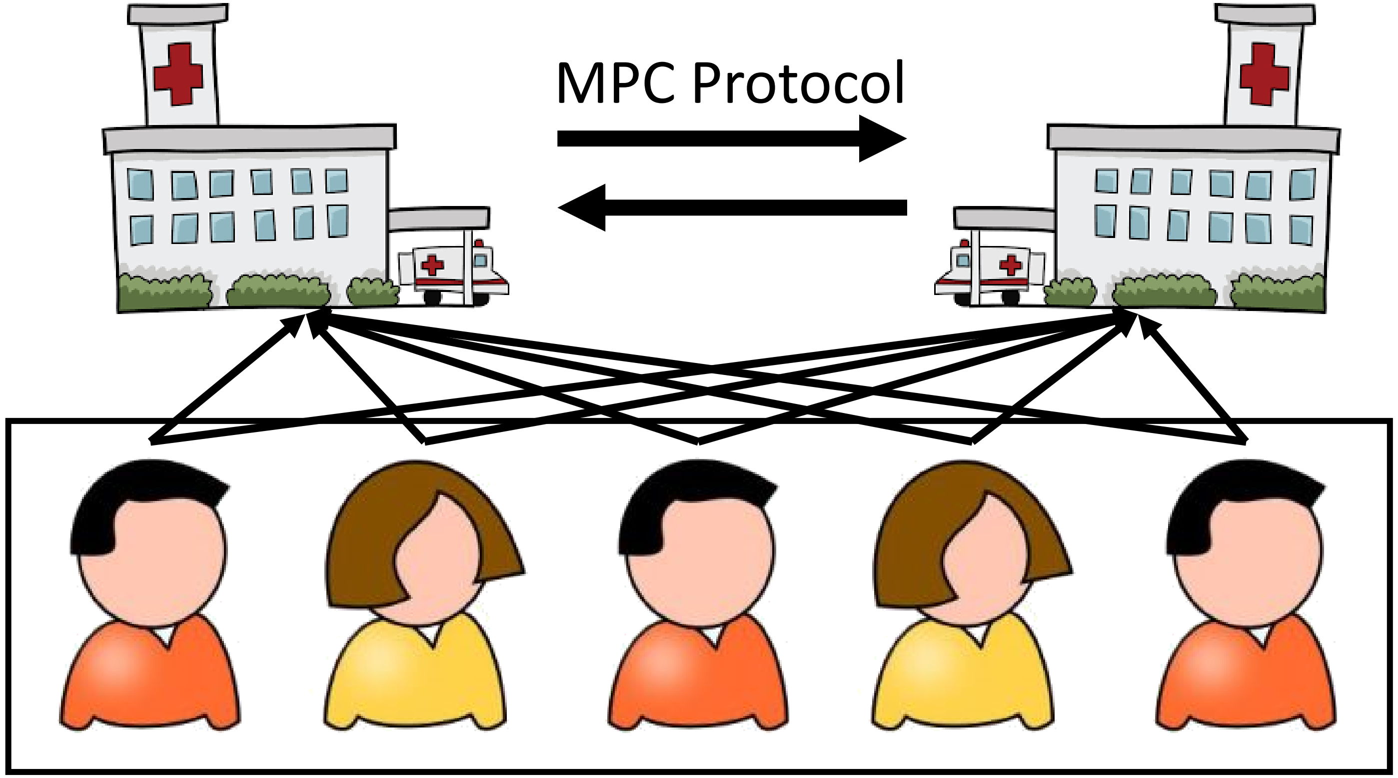
For computing products on shared values (e.g., matrix-vector products, inner products, etc.), we can use a single-round interactive protocol [Bea91]

0
1
⋮
-7

$[v_1]_2 + [v_2]_2 = [v_1 + v_2]_2$

What About More Complex Diseases?

[CWB18]



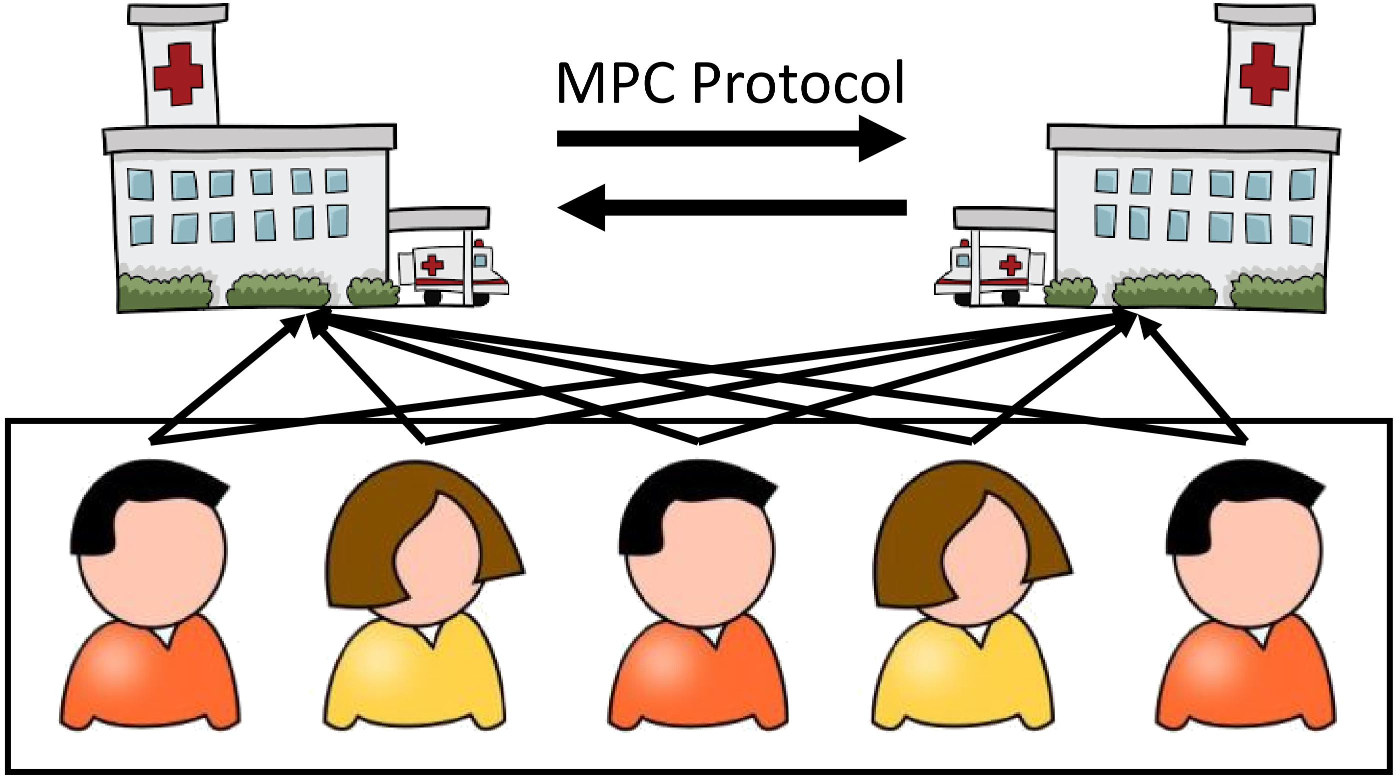
Approach: directly compute on secret-shared data

This work: first end-to-end GWAS protocol (with population correction)

- Based on computing on secret-shared inputs
- For 25K individuals, computation completes in about 3 days: feasible for performing large-scale scientific studies

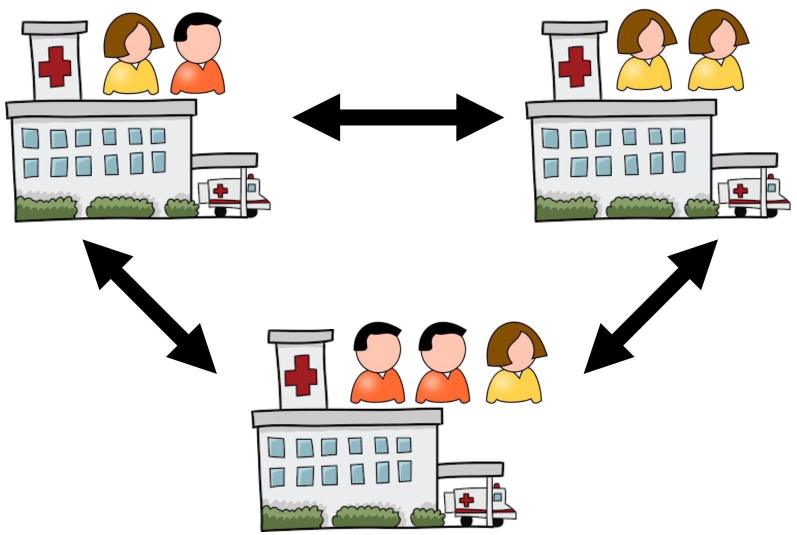
Can compose with differential privacy to ensure outputs preserve privacy of user data

Secure Genome Computation

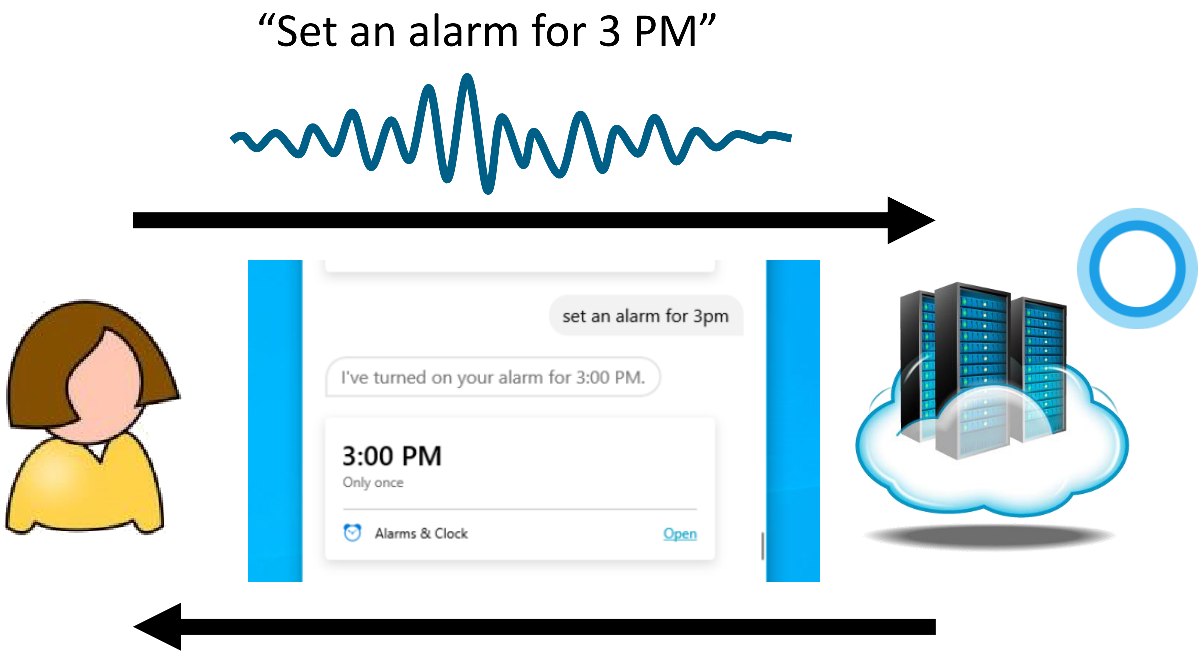


Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes

Privacy-Preserving Machine Learning



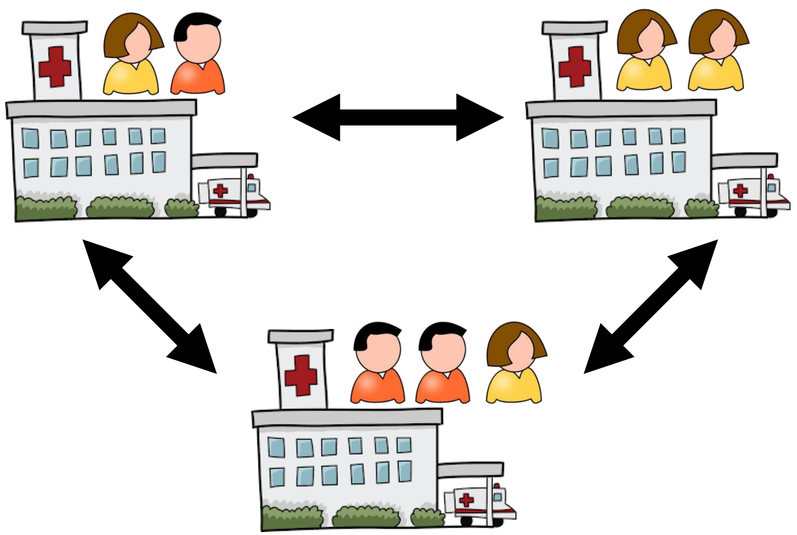
Private training: Multiple parties train a joint model on their aggregate data while ensuring <u>privacy</u> of the input data

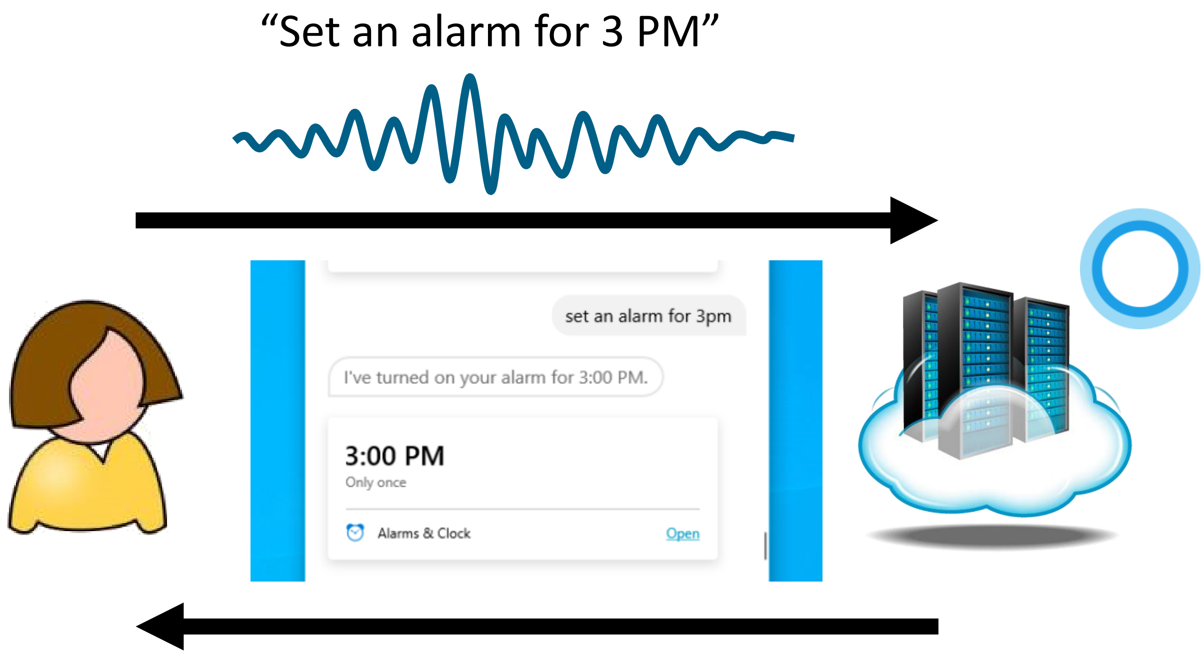


Private inference: Client learns model's output, server does not learn anything

Machine learning as a service

Privacy-Preserving Machine Learning



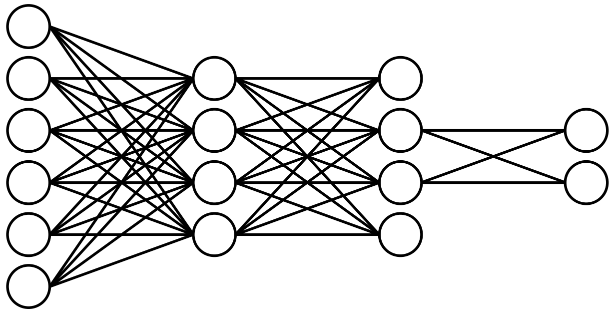


Many constructions in recent years:

MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY^3 [MR18], Chameleon [RWTSS+18], SecureNN [WGC19], XONN [RSCLL+19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MLSZP20], FLASH [BCPS20], Trident [CRS20], CrypTFlow [KRCDR+20], Falcon [WTBKM+21]

Privacy-Preserving Machine Learning

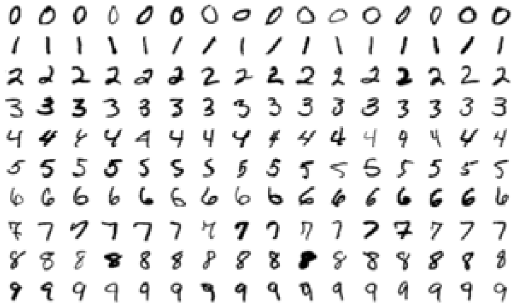
Simple models and datasets:



Feed-forward neural networks

10,000 – 100,000 parameters

2 – 3 layers



MNIST dataset

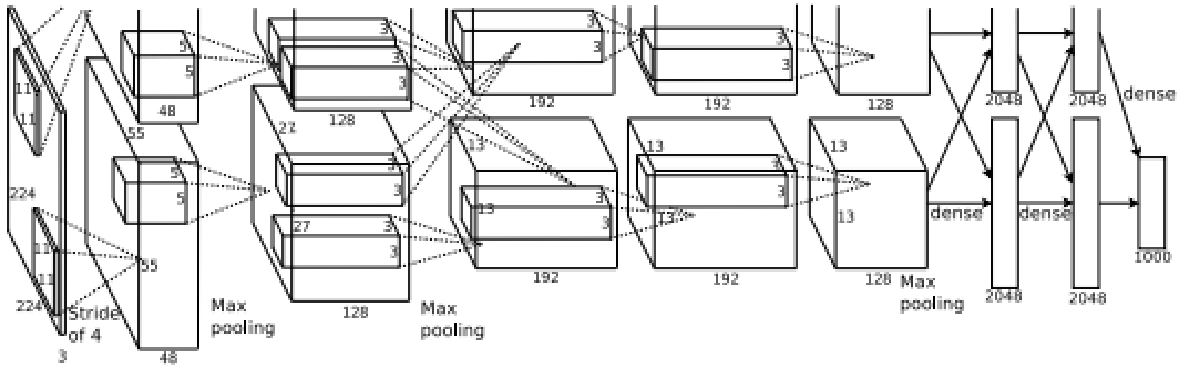
10 classes; 60,000 examples

Many constructions in recent years:

MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY^3 [MR18], Chameleon [RWTSS+18], SecureNN [WGC19], XONN [RSCLL+19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MLSZP20], FLASH [BCPS20], Trident [CRS20], CrypTFlow [KRCDR+20], Falcon [WTBKM+21]

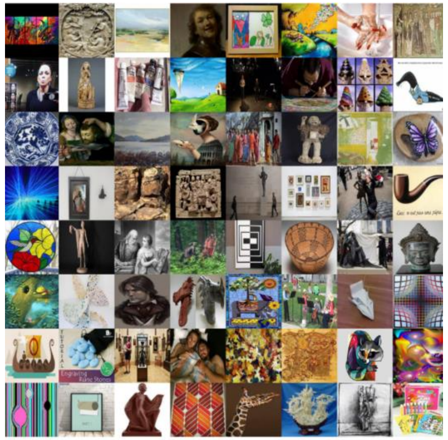
Privacy-Preserving Machine Learning

Larger models and datasets:



AlexNet [KSH12]

61,000,000 parameters
8 layers



Tiny ImageNet [LKJ17]

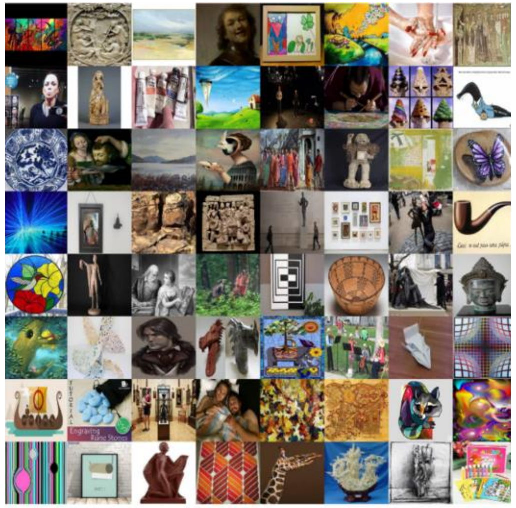
Subset of ImageNet [RDSKS+15]
200 classes, 100,000 examples

Many constructions in recent years:

MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY^3 [MR18], Chameleon [RWTSS+18], SecureNN [WGC19], XONN [RSCLL+19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MLSZP20], FLASH [BCPS20], Trident [CRS20], CrypTFlow [KRCDR+20], Falcon [WTBKM+21]

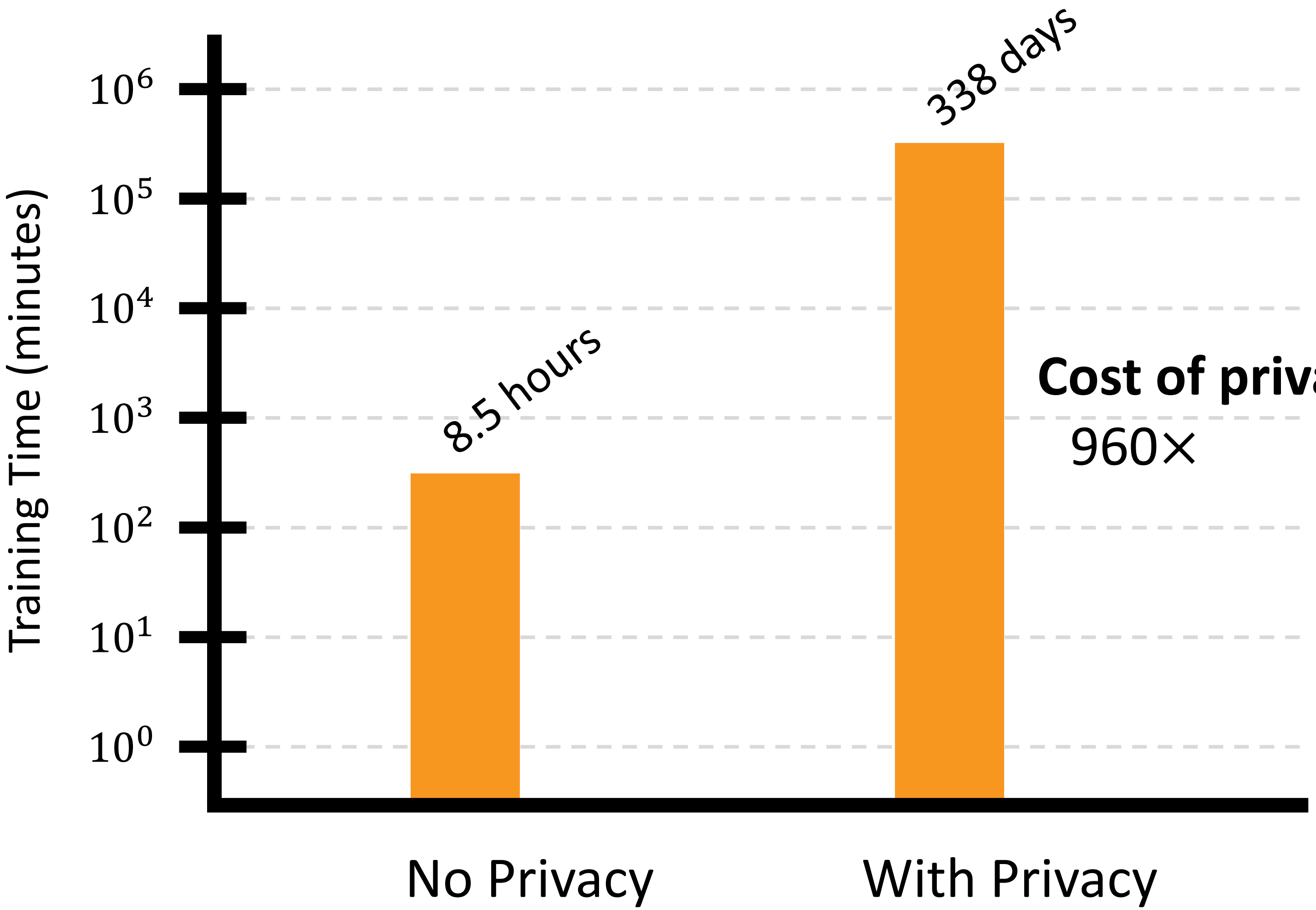
The Scalability Challenge in Private ML

Tiny ImageNet [LKJ17]



Subset of ImageNet [RDSKS+15]
200 classes, 100,000 examples

Training the AlexNet model on Tiny ImageNet



Cost of privacy:
960×

[Assuming 90 epochs over dataset]

Modern Deep Learning (without Privacy)

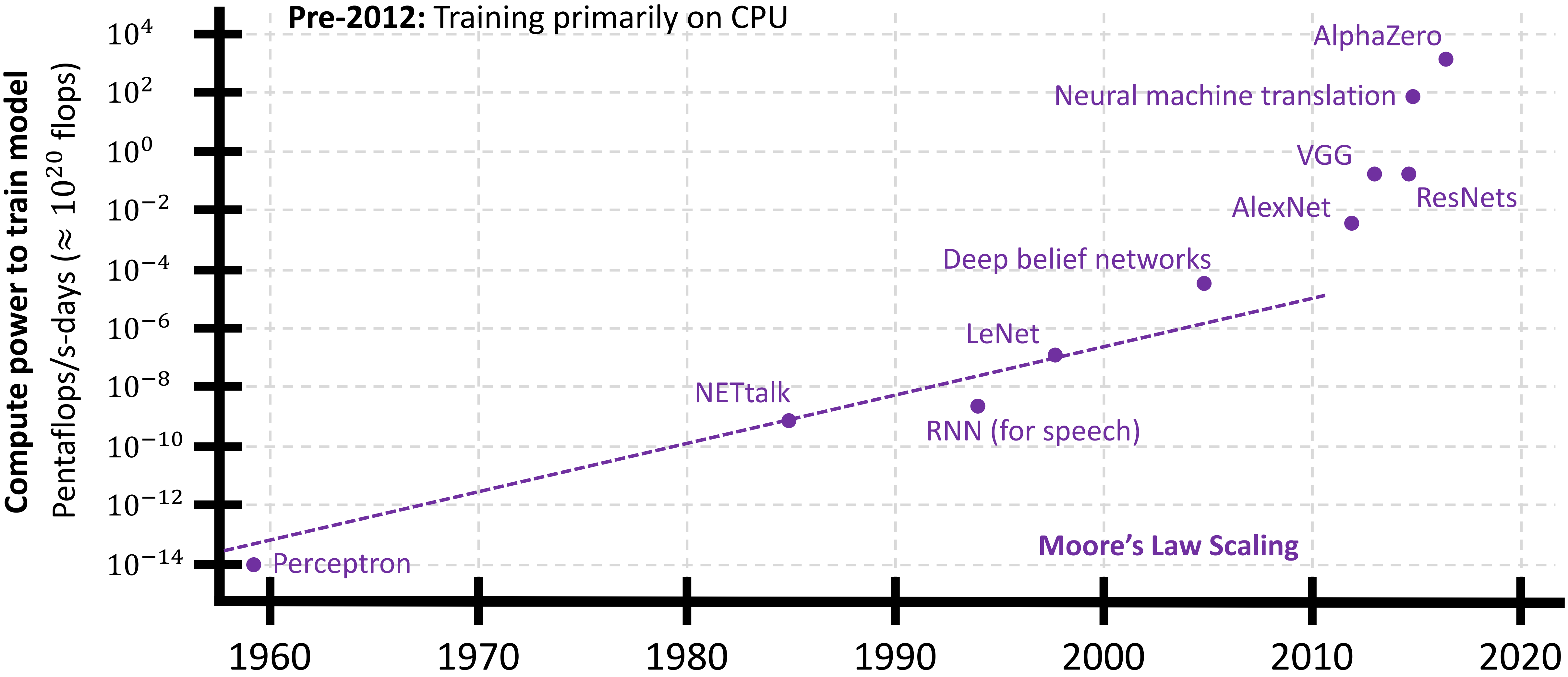


Figure adapted from OpenAI blog post

Modern Deep Learning (without Privacy)

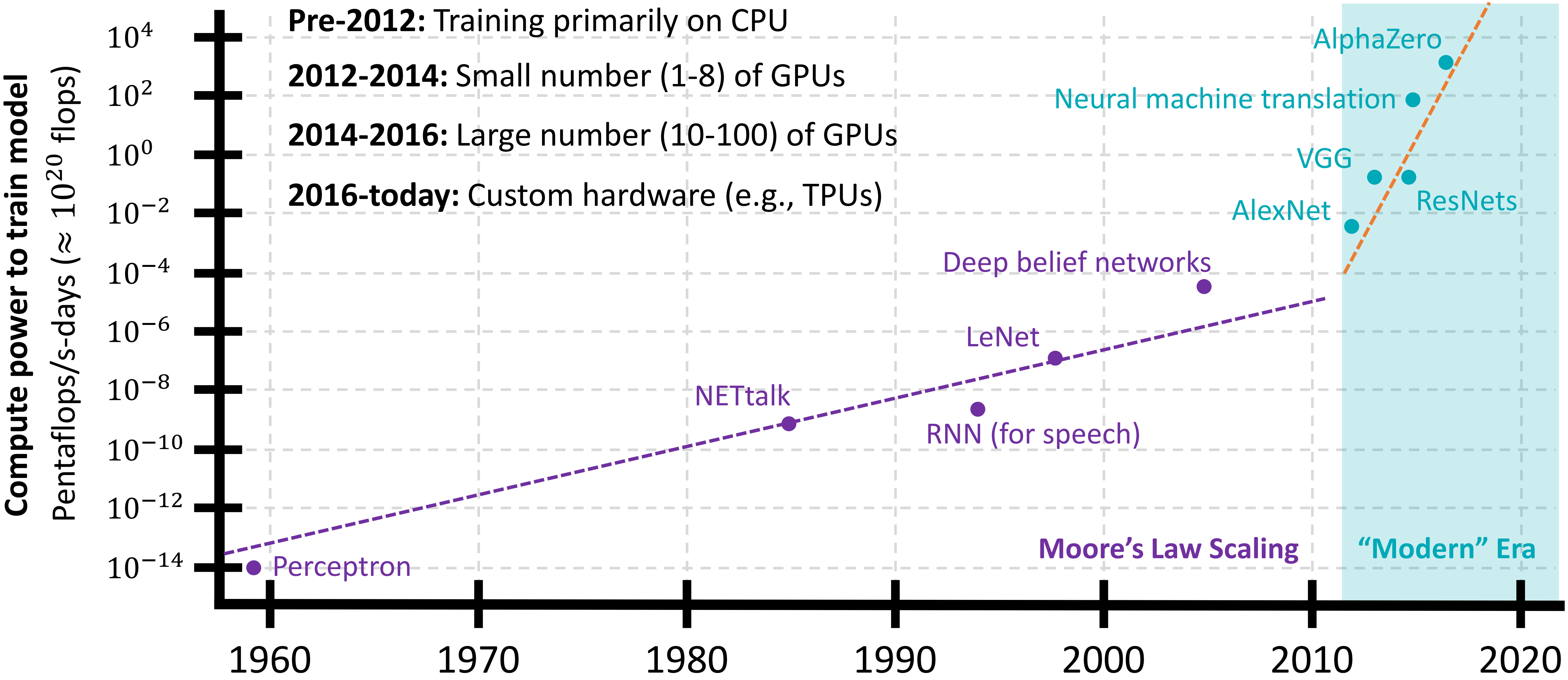


Figure adapted from OpenAI blog post

CryptGPU: Private ML on the GPU

[TKTW21]

First cryptographic framework where all computations performed on the GPU

Supporting Cryptography on the GPU:

Cryptographic protocols designed for general-purpose computing architectures

CUDA kernels for linear algebra operate on floating-point types while cryptographic protocols operate on discrete data types (e.g., finite fields)

This Work:

New abstractions and protocols to embed cryptographic operations onto the GPU

CryptGPU: Private ML on the GPU

[TKTW21]

First cryptographic framework where all computations performed on the GPU

Motivates the study of **"GPU-friendly" cryptography**

Supporting Cryptography on the GPU:

Cryptographic protocols designed for general-purpose computing architectures

CUDA kernels for linear algebra operate on floating-point types while cryptographic protocols operate on discrete data types (e.g., finite fields)

This Work:

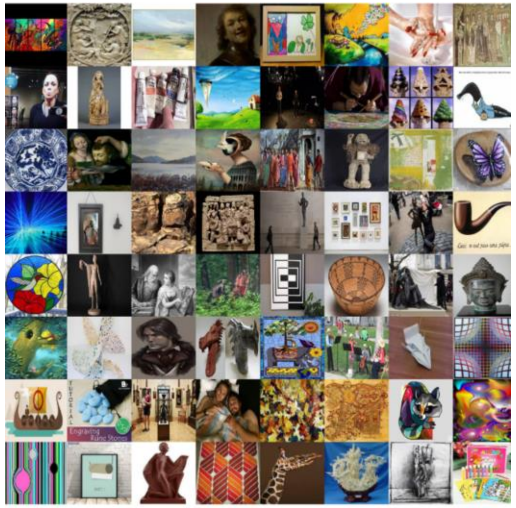
New abstractions and protocols to embed cryptographic operations onto the GPU

New protocols to take better advantage of GPU acceleration

CryptGPU: Private ML on the GPU

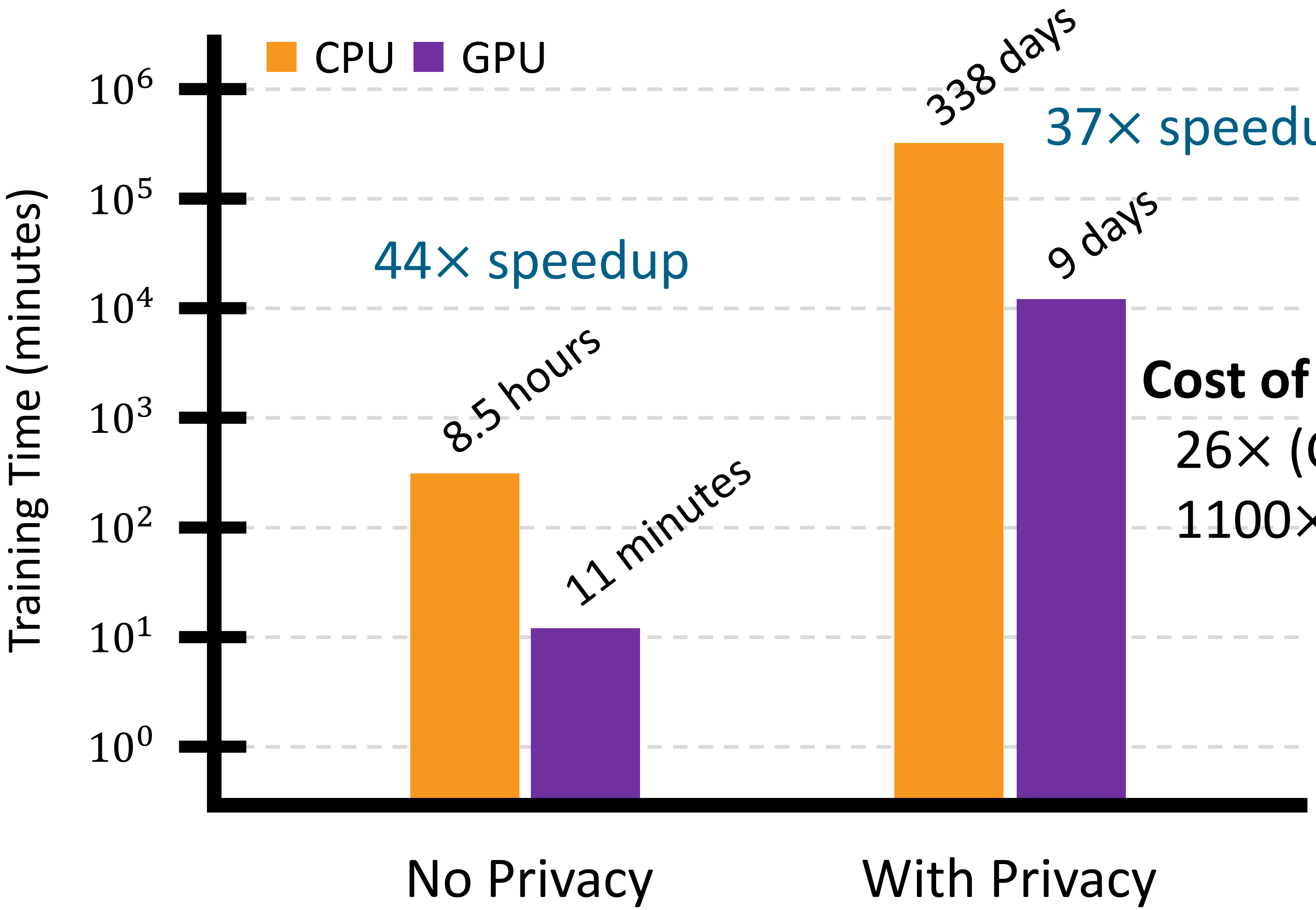
[TKTW21]

For private training, largest dataset to date is Tiny ImageNet [LKJ17]



Subset of ImageNet [RDSKS+15]
200 classes, 100,000 examples

Training the AlexNet model on Tiny ImageNet



Cost of privacy:
26× (CPU)
1100× (GPU)

[Assuming 90 epochs over dataset]

Towards an AlexNet Moment for Private ML

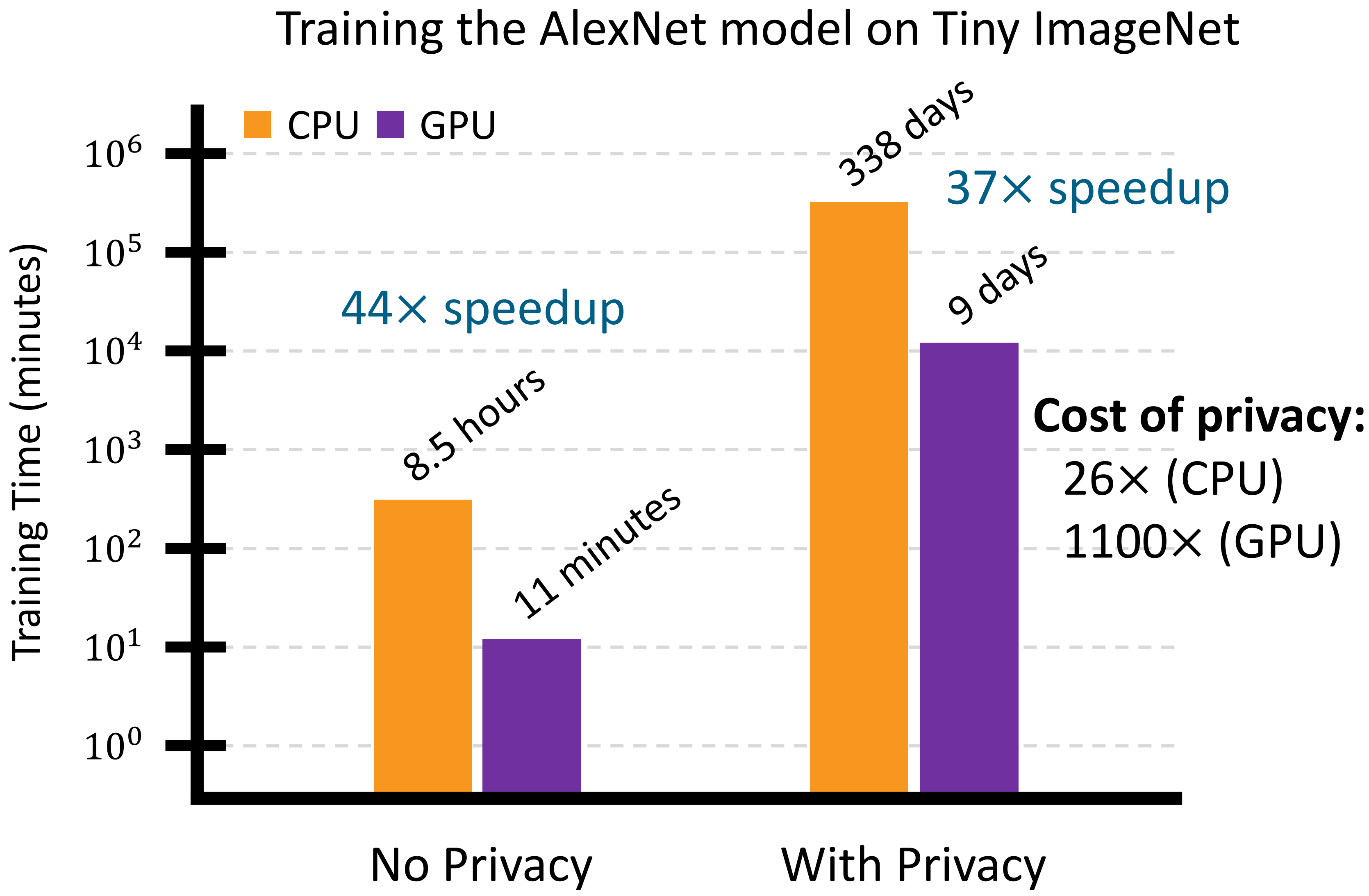
[TKTW21]

Feasibility result: Possible to run cryptographic protocols entirely on the GPU

More improvements possible if we tailor protocol design to GPU architecture

- Specialized CUDA kernels for cryptographic operations
- New embeddings for discrete cryptographic structures

Can we take advantage of even more specialized hardware (FPGAs, TPUs, etc.)?



Computing on Private Data

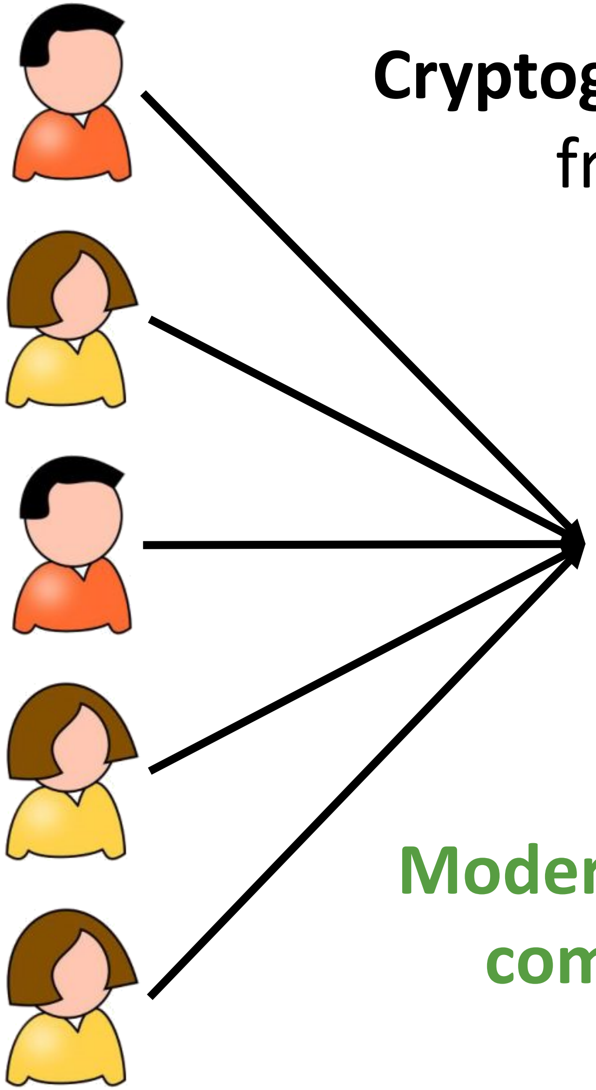
x_1

x_2

x_3

x_4

x_5



Cryptographic tools: hide the input x_i from the computing party

Complementary goals as differential privacy (and oftentimes, can be *combined*)

Aggregate statistic
$y \leftarrow f(x_1, \ldots, x_5)$

Modern cryptographic tools enable computations on private data

Thank you!