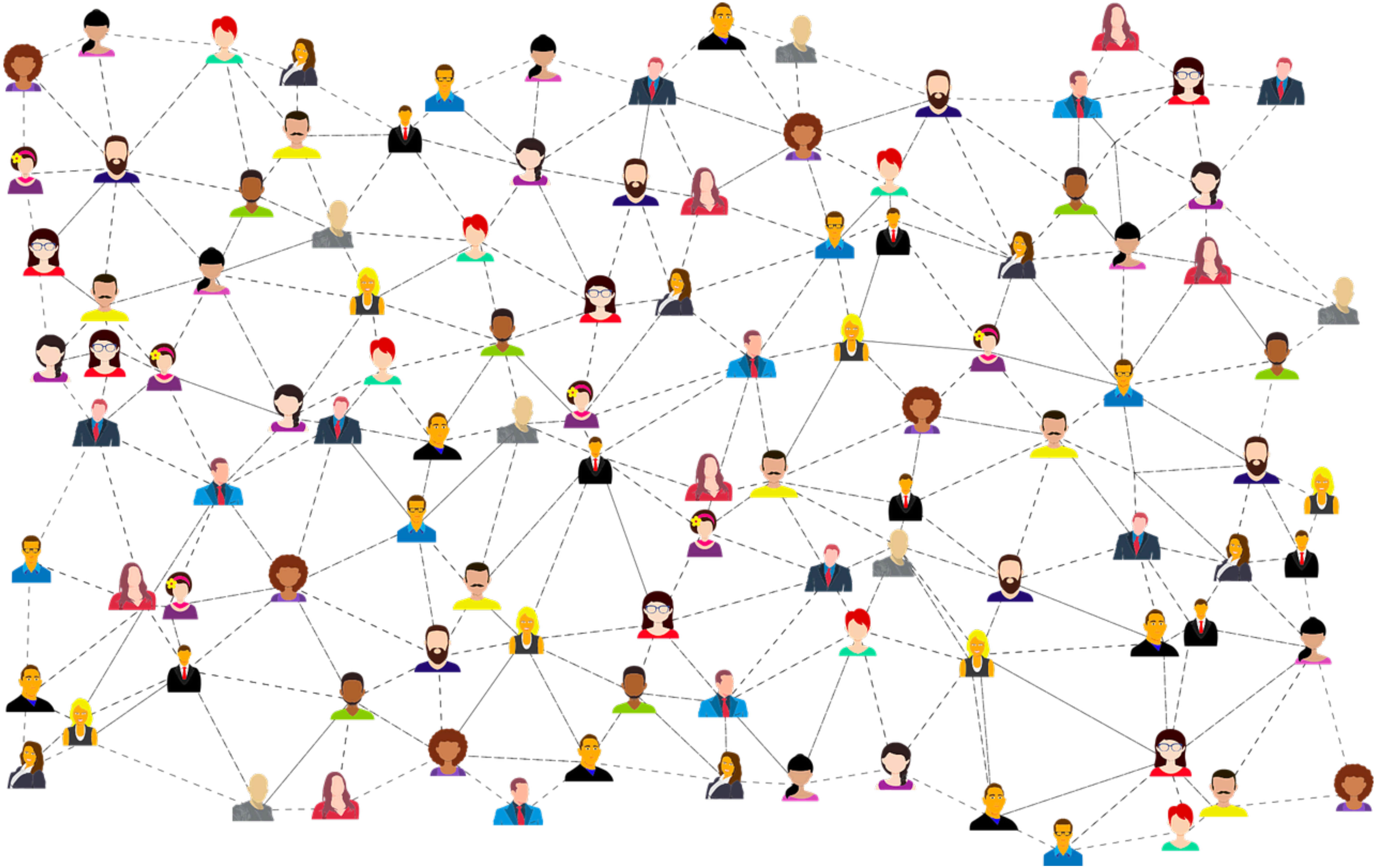
Secure Computation

Protecting the Privacy of Data used in Distributed Computation

Geoffroy Couteau

Université de Paris

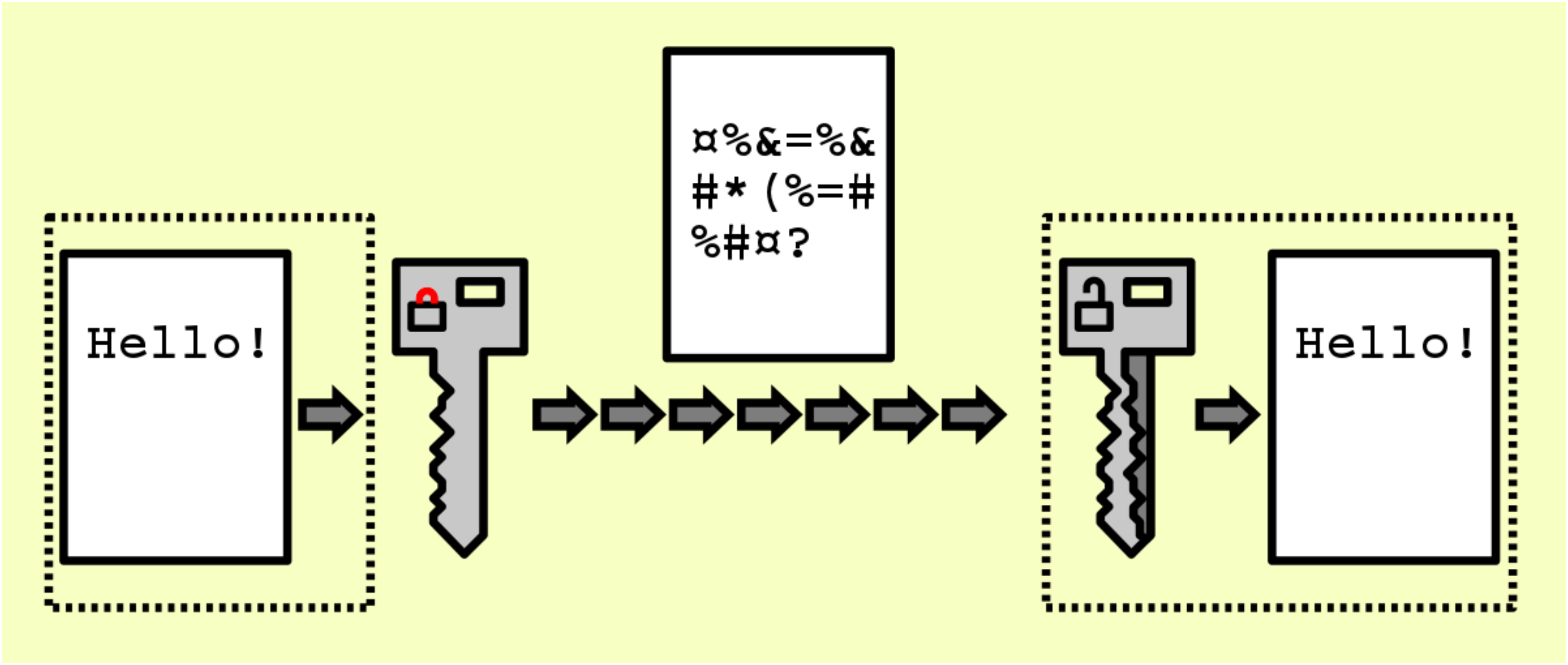
Are our Interactions over Large Networks Secure?



Are our Interactions over Large Networks Secure?

Our *Communications* are Mostly* Secure

Whenever we browse the web, use a website or an app, send a message, or make a call, we **communicate over a network**, and the content of our communication is private information. Most of the time*, this communication happens **securely**:



- Since 2020, **around 85% of the total internet traffic** is encrypted
- End-to-end encryption is becoming a standard on most messaging apps
- Cellular networks in France encrypt **all communications** by default

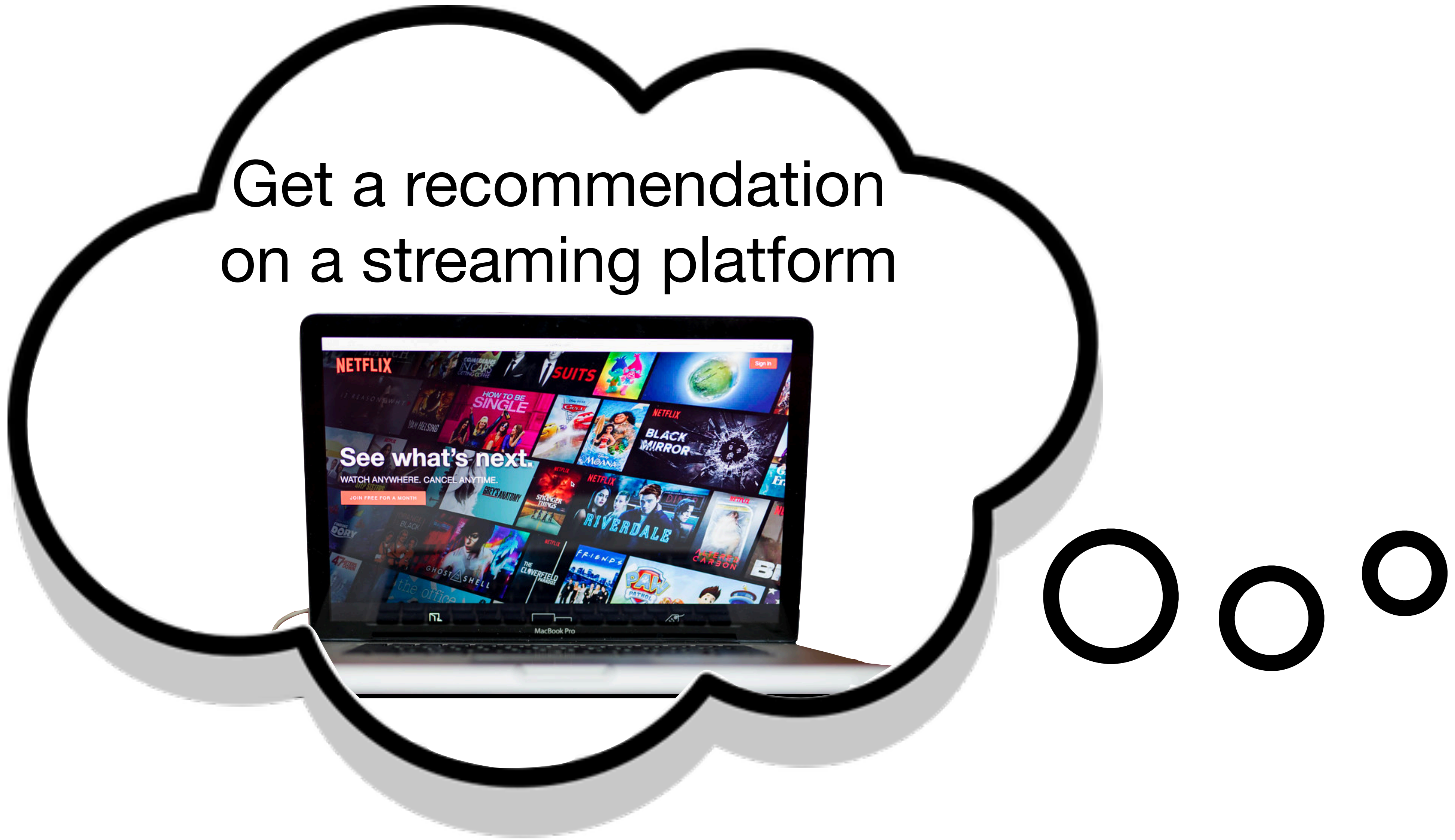
** not always!*

But our *Computations* are not!

Our use of networks has evolved: whenever we…

But our *Computations* are not!

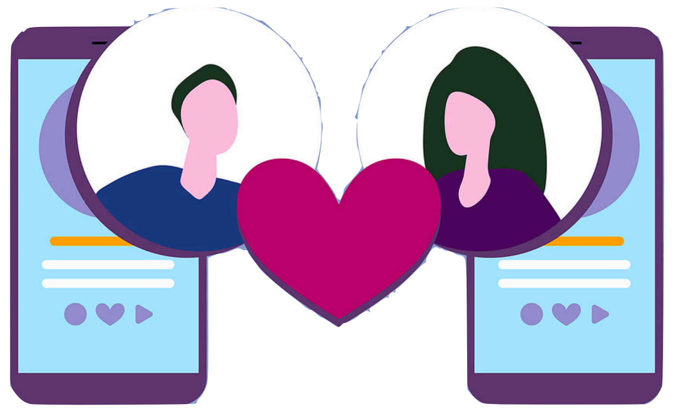
Our use of networks has evolved: whenever we…



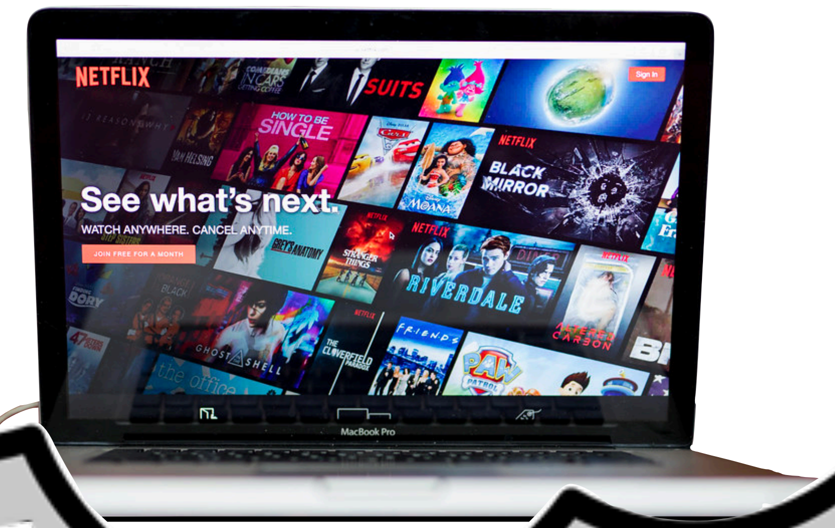
But our *Computations* are not!

Our use of networks has evolved: whenever we…

Use a dating app

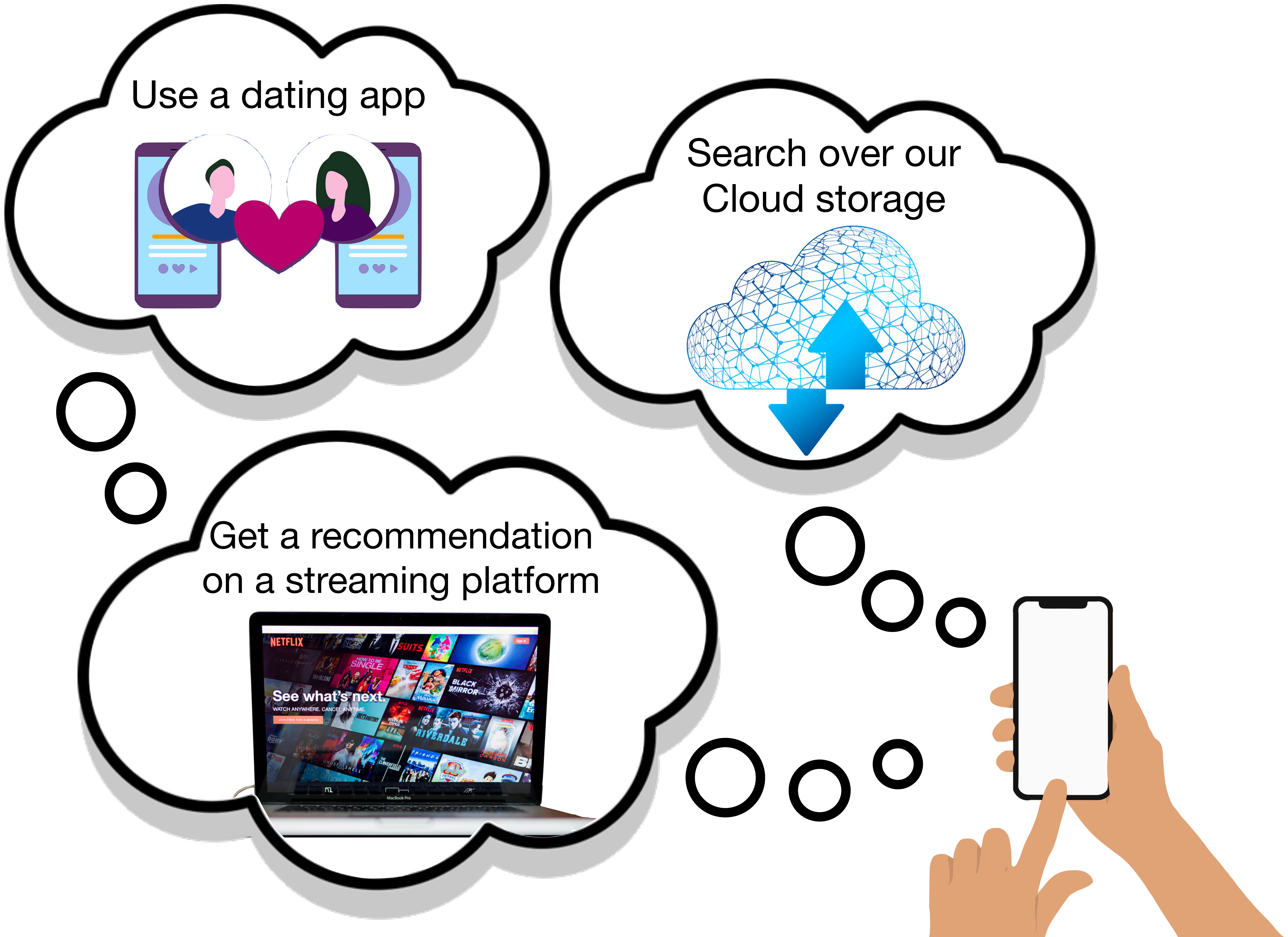


Get a recommendation on a streaming platform



But our *Computations* are not!

Our use of networks has evolved: whenever we…

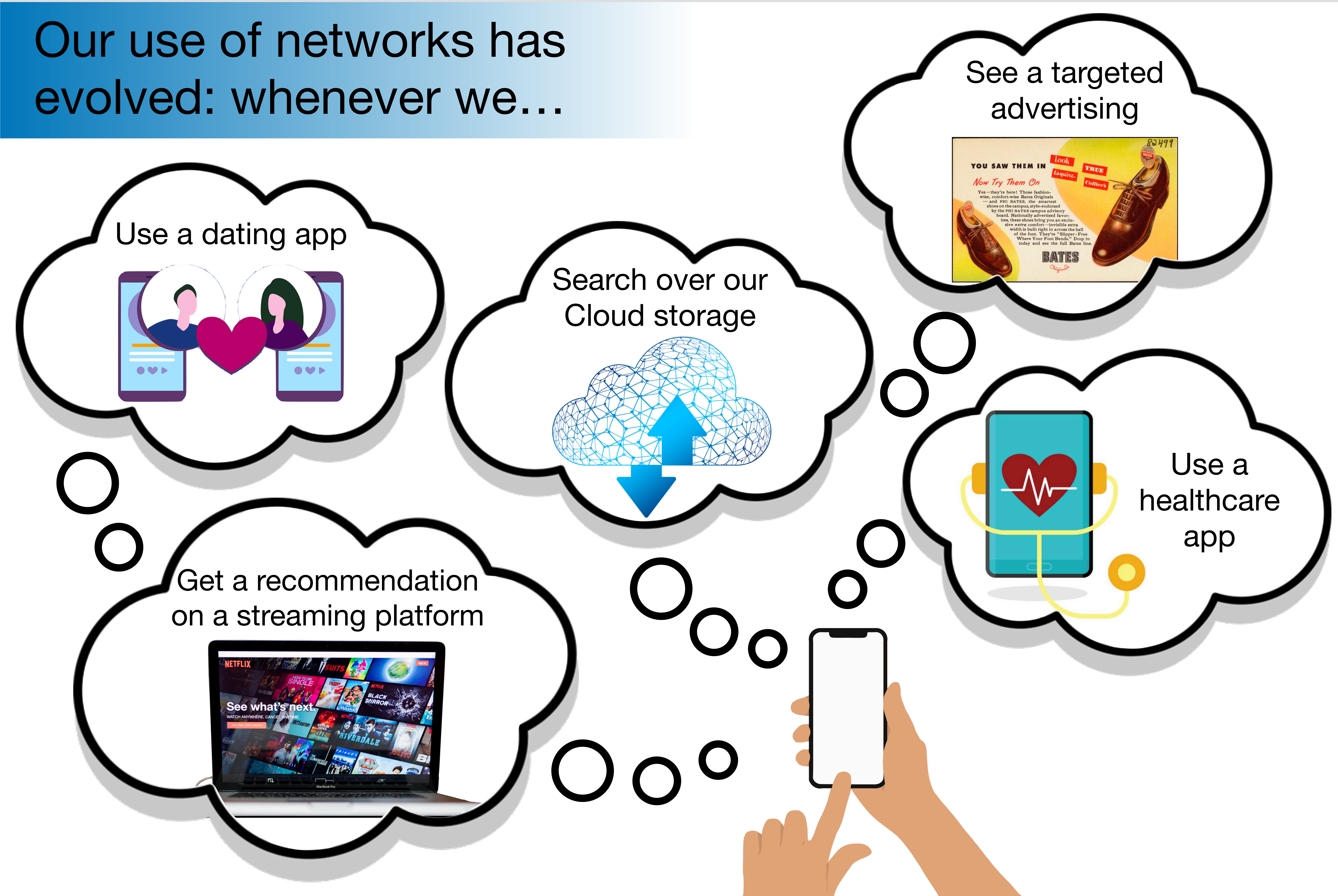


But our *Computations* are not!

Our use of networks has evolved: whenever we…

But our *Computations* are not!

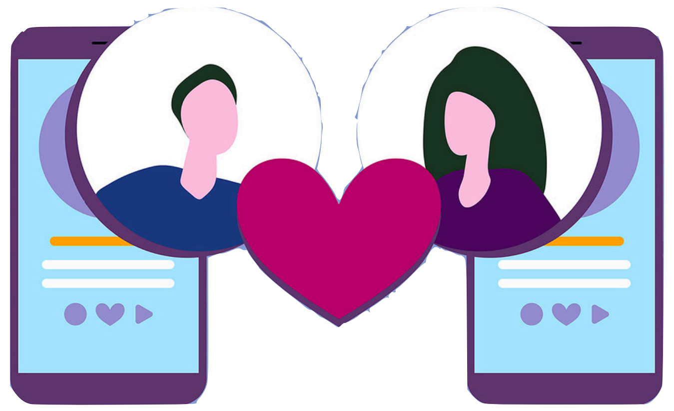
Our use of networks has evolved: whenever we…



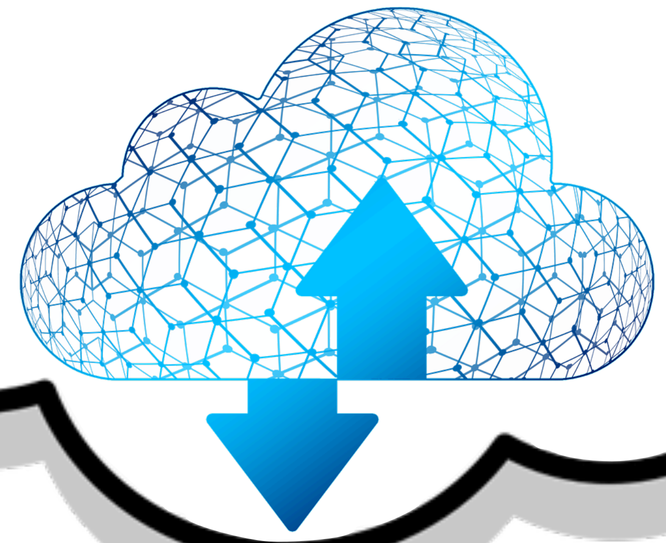
But our *Computations* are not!

Our use of networks has evolved: whenever we…

Use a dating app

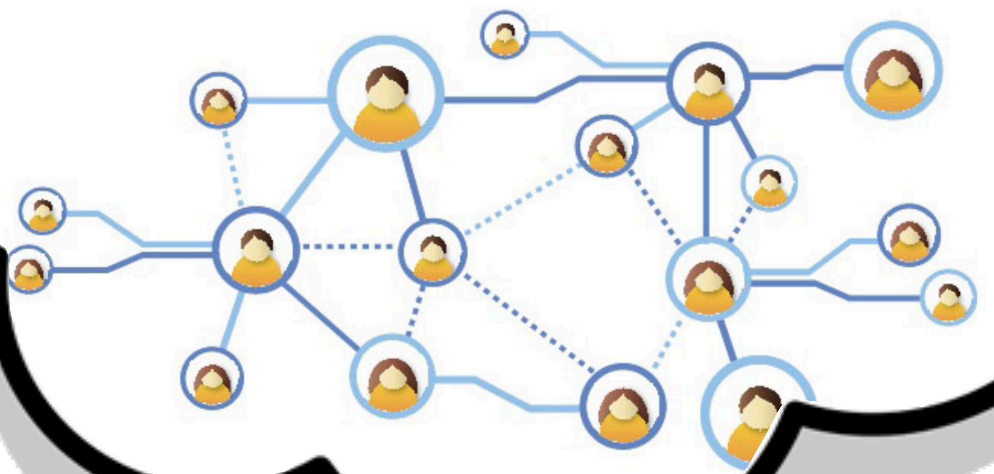


Search over our Cloud storage



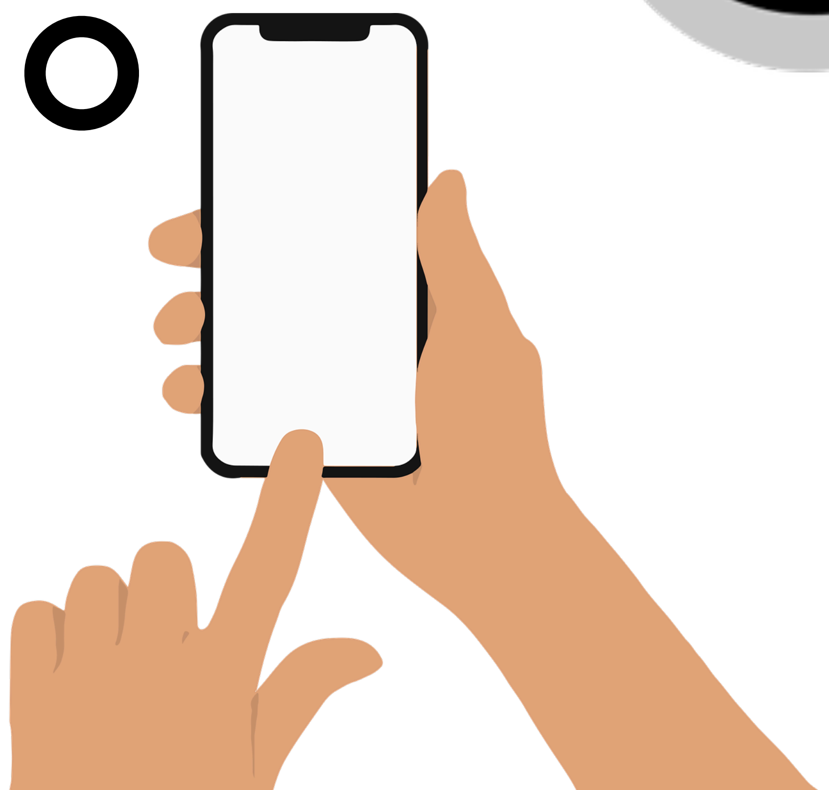
See a targeted advertising

Use a social network



Use a healthcare app

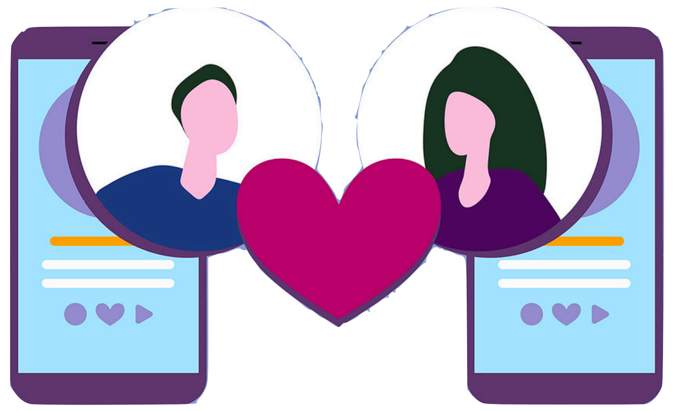
Get a recommendation on a streaming platform



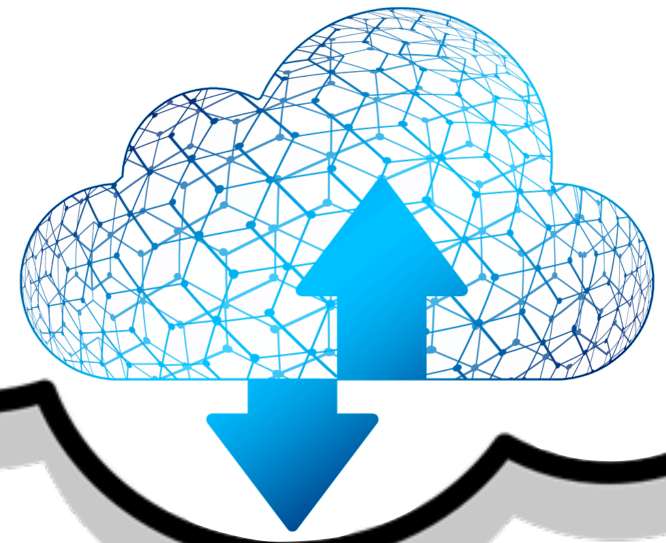
But our *Computations* are not!

Our use of networks has evolved: whenever we…

Use a dating app

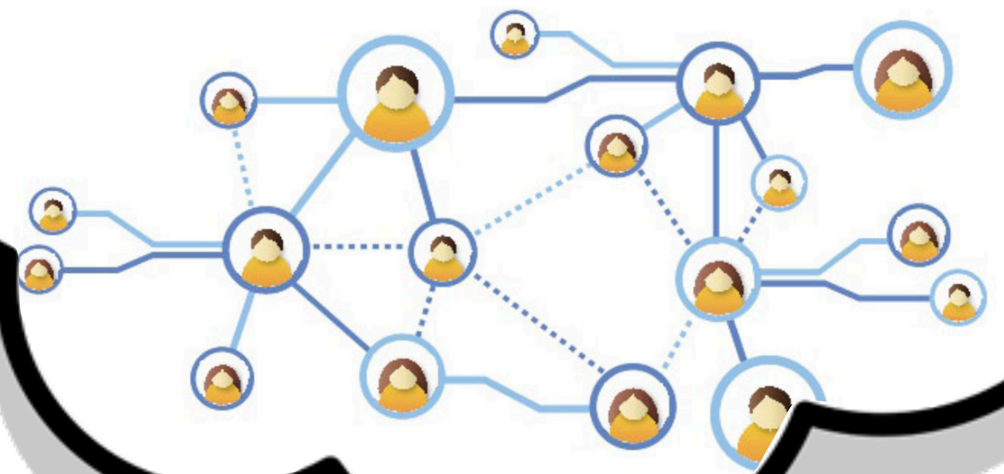


Search over our Cloud storage



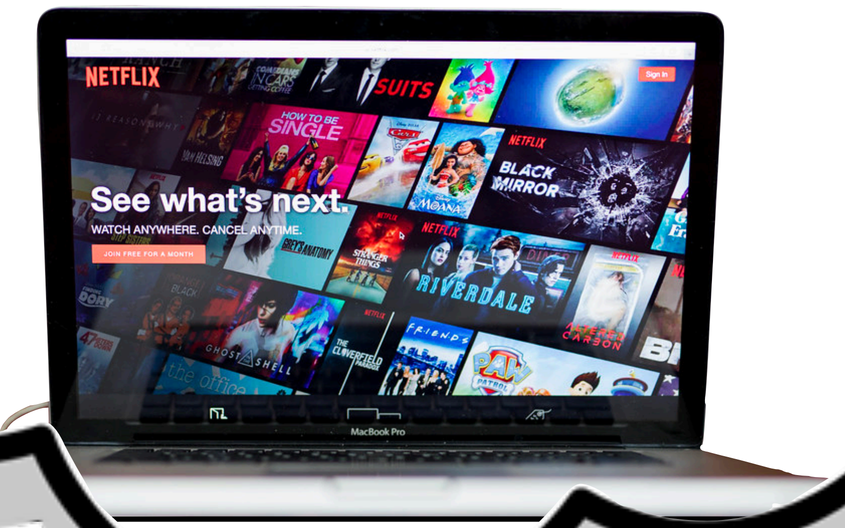
See a targeted advertising

Use a social network

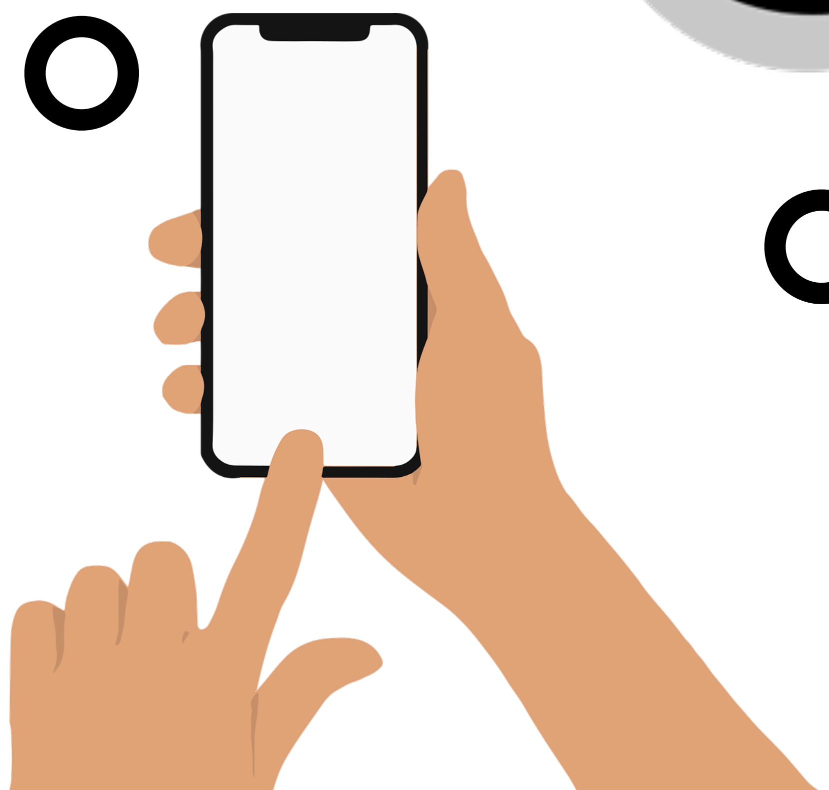


Use a healthcare app

Get a recommendation on a streaming platform



…



But our *Computations* are not!

Our use of networks has evolved: whenever we…

Our private data is **used in computations**

A Paradoxical Situation

We become increasingly aware of the **need for privacy** in communications

- Over the web
- When using messaging apps

We are strongly **incentivized** to distribute our private data

- To benefit from AI-driven apps (photos,  health apps…)
- To use social networks (friend recommendations, curated timelines…)

And our data is becoming **extremely valuable**

- For targeted advertising
- To train machine learning algorithms (e.g. to find new treatments)

As a result, we protect our privacy whenever we **communicate**, but give up on it whenever **computations** are required… Which happens on a daily basis.

A Paradoxical Situation

We become increasingly aware of the **need for privacy** in communications
- Over the web
- When using messaging apps

We are strongly **incentivized** to distribute our private data
- To benefit from AI-driven apps (photos, health apps…)
- To use social networks (friend recommendations, curated timelines…)

And our data is becoming **extremely valuable**
- For targeted advertising
- To train machine learning algorithms (e.g. to find new treatments)

⚠ The solution is **not** to « tell users to be careful ». It is unrealistic:

- To hope that users will stop using apps and social networks, and
- To give up on societal benefits of computations on private data.

A Paradoxical Situation

We become increasingly aware of the **need for privacy** in communications

- Over the web
- When using messaging apps

We are strongly **incentivized** to distribute our private data

- To benefit from AI-driven apps (photos, health apps…)
- To use social networks (friend recommendations, curated timelines…)

And our data is becoming **extremely valuable**

- For targeted advertising
- To train machine learning algorithms (e.g. to find new treatments)

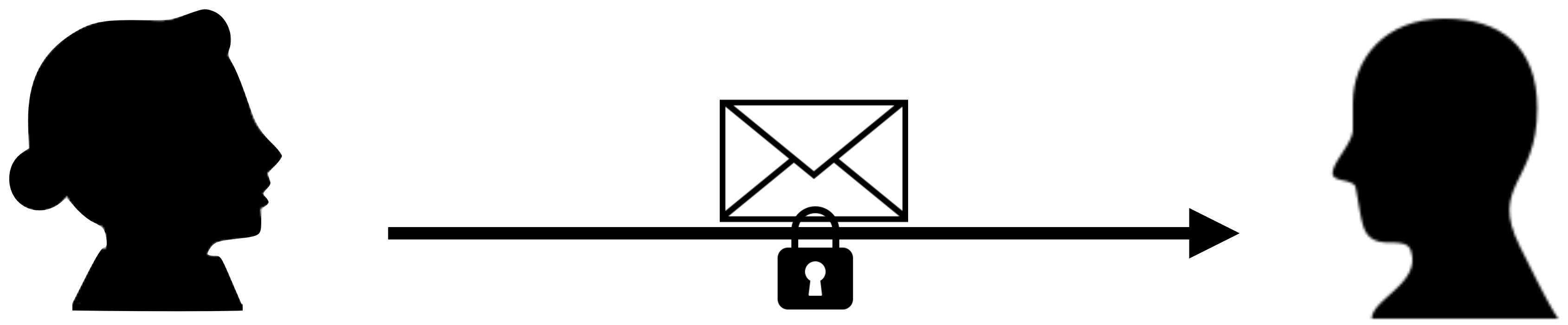
Secure computation aims to reconcile the (individual, societal) **benefits** of computations on data with the need to **protect its privacy**.

What is Secure Computation?

Protecting traditional uses of networks

Secure communication

Goal: *communicating* a secret message



m

Output: Bob learns m

Security: Eve learns nothing

Solved by **encryption**

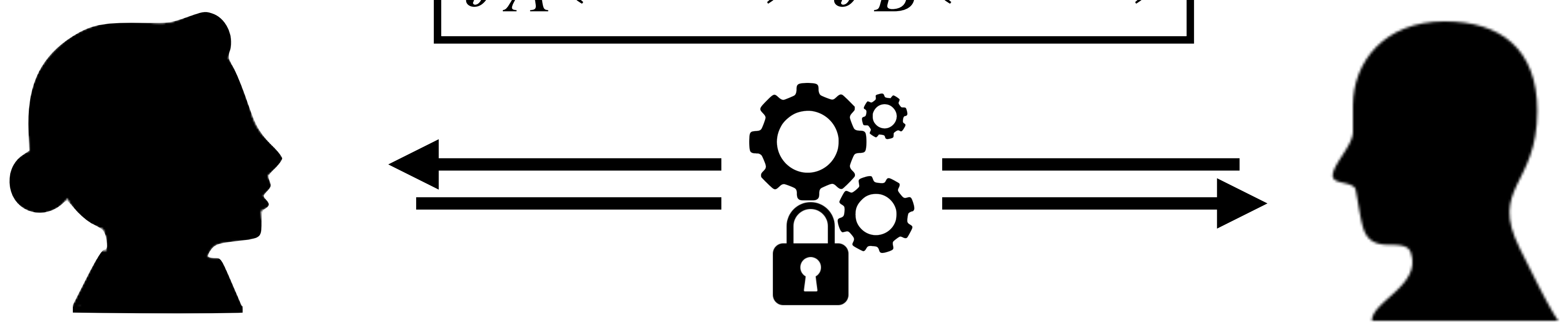
Locks the message in a digital « box »
Only the owner of the key can read it

Protecting modern uses of networks

Secure computation

Goal: *computing* (public) functions on secret inputs

$f_A(\cdot,\cdot), f_B(\cdot,\cdot)$



x y

Output: Alice learns $f_A(x,y)$ and Bob learn $f_B(x,y)$

Security: Alice and Bob learn nothing else

Encryption is « all or nothing »

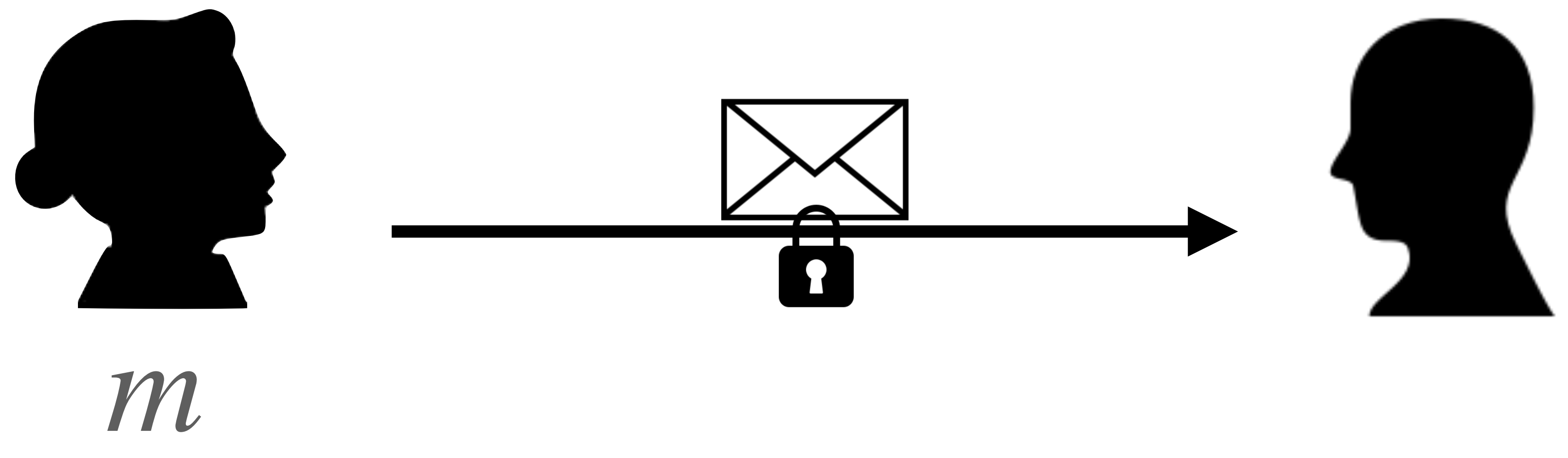
It does not allow a *fine-grained* access to some *specific* information about the data

What is Secure Computation?

Protecting traditional uses of networks

Secure communication

Goal: *communicating* a secret message



Output: Bob learns m

Security: Eve learns nothing

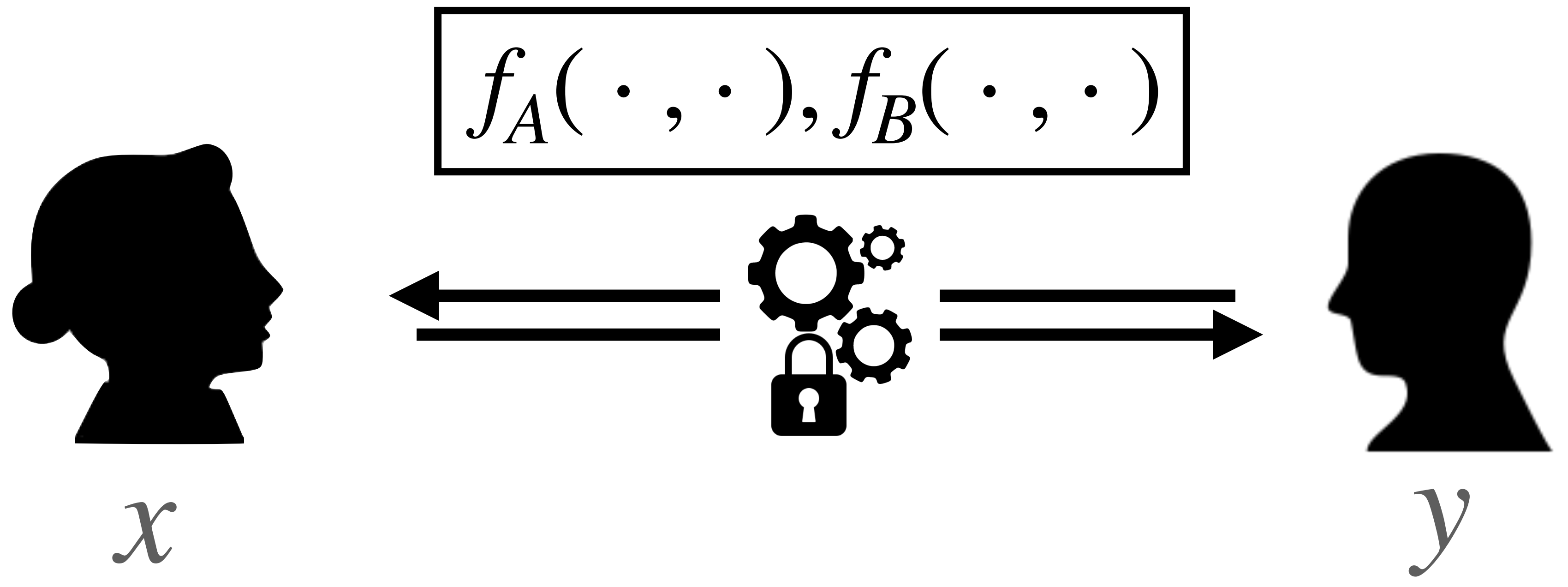
Solved by **encryption**

Locks the message in a digital « box »
Only the owner of the key can read it

Protecting modern uses of networks

Secure computation

Goal: *computing* (public) functions on secret inputs



Output: Alice learns $f_A(x,y)$ and Bob learn $f_B(x,y)$

Security: Alice and Bob learn nothing else

Encryption is « all or nothing »

It does not allow a *fine-grained* access to some *specific* information about the data

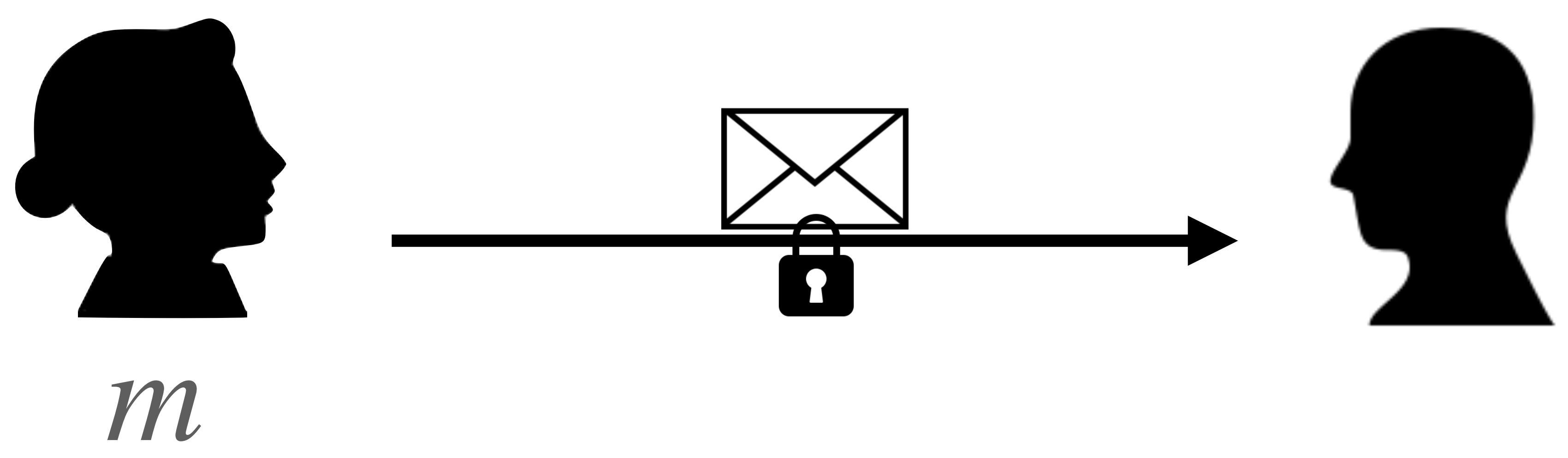
Secure computation is the area of security that studies techniques and protocols to allow computing public functions on *private* inputs

What is Secure Computation?

Protecting traditional uses of networks

Secure communication

Goal: *communicating* a secret message



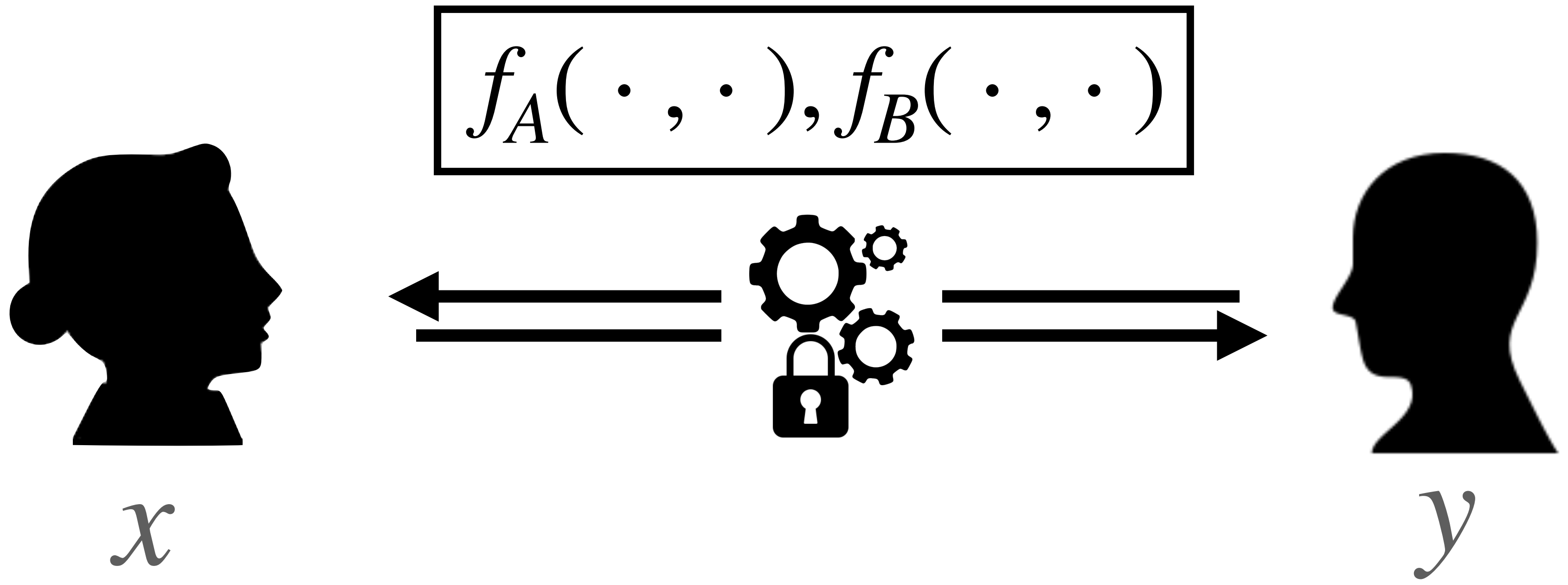
Output: Bob learns m

Security: Eve learns nothing

Protecting modern uses of networks

Secure computation

Goal: *computing* (public) functions on secret inputs

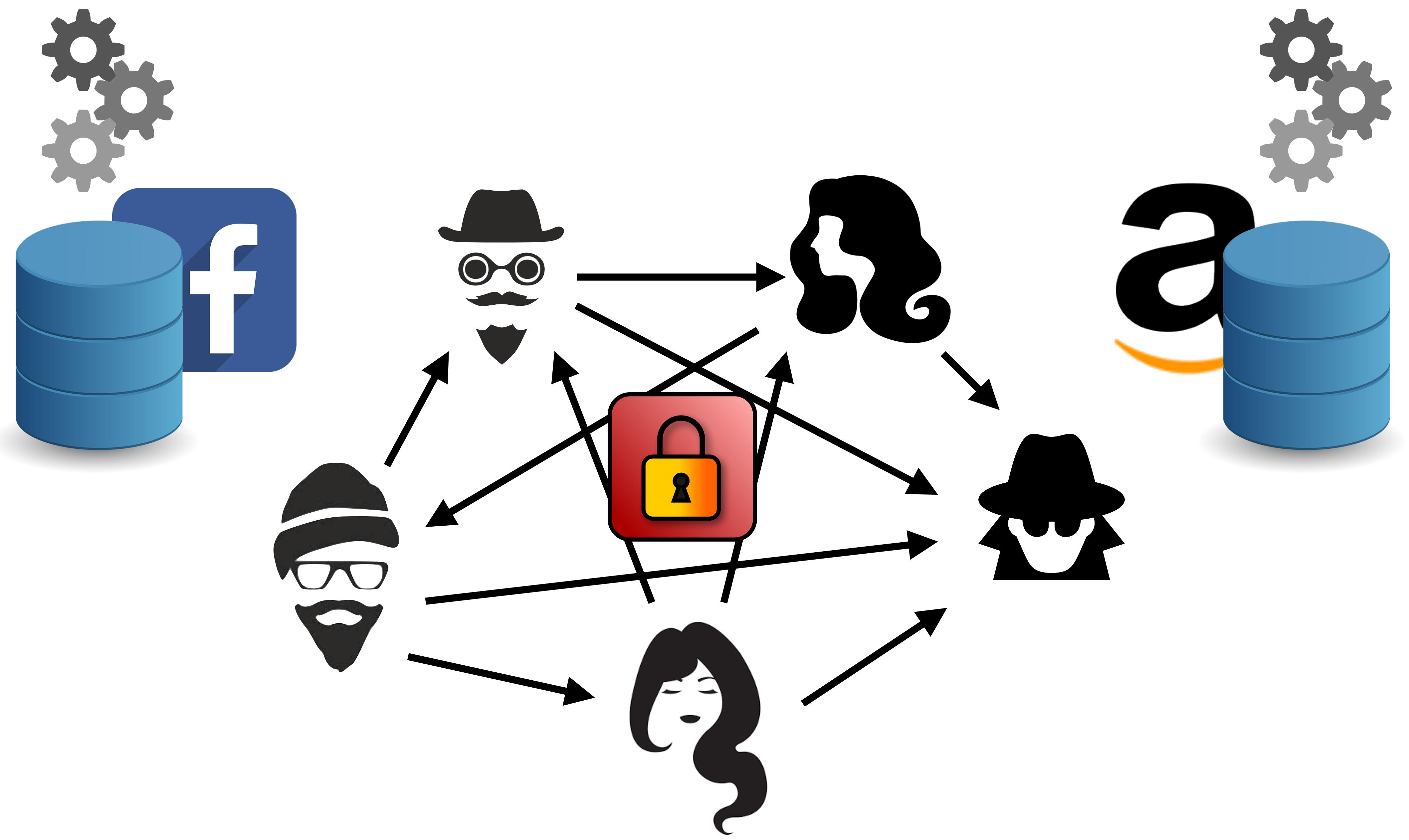


Output: Alice learns $f_A(x, y)$ and Bob learn $f_B(x, y)$

Security: Alice and Bob learn nothing else

- Secure computation is a more *fine-grained* approach to security: the function controls precisely what is learned (secure communication is *all or nothing*)
- It is much more demanding: now the adversary is *internal* (Alice must be protected against Bob, and Bob against Alice), and can influence the protocol!

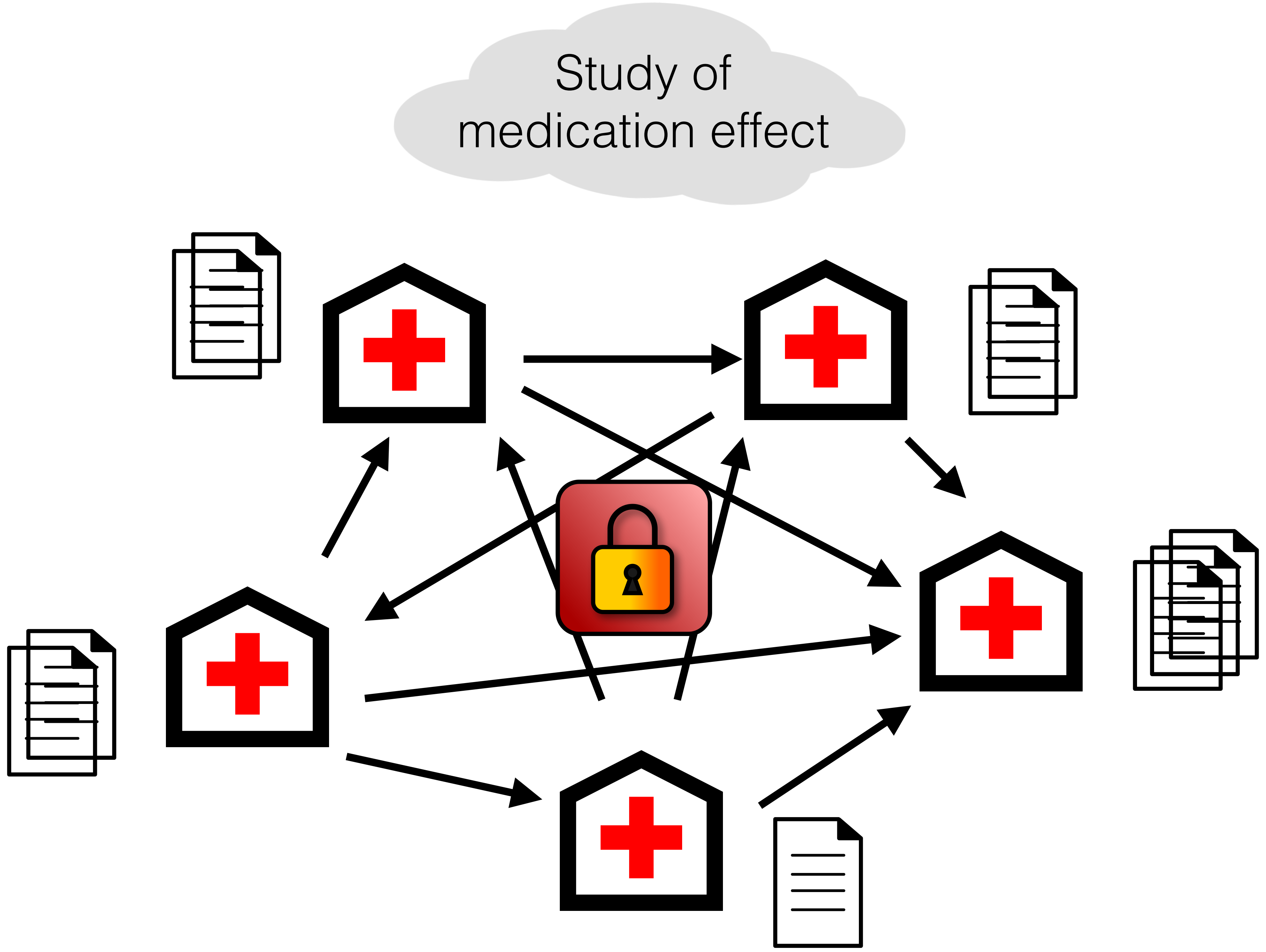
What is Secure Computation?



More generally, n participants $P_1, \cdots, P_n$ with private inputs $x_1, \cdots, x_n$ wish to distributively compute $(y_1, \cdots, y_n) \leftarrow f(x_1, \cdots, x_n)$ such that

- **Correctness:** at the end of the interaction, P_i learns y_i
- **Security:** no *coalition of parties* learns anything beyond their own inputs and outputs

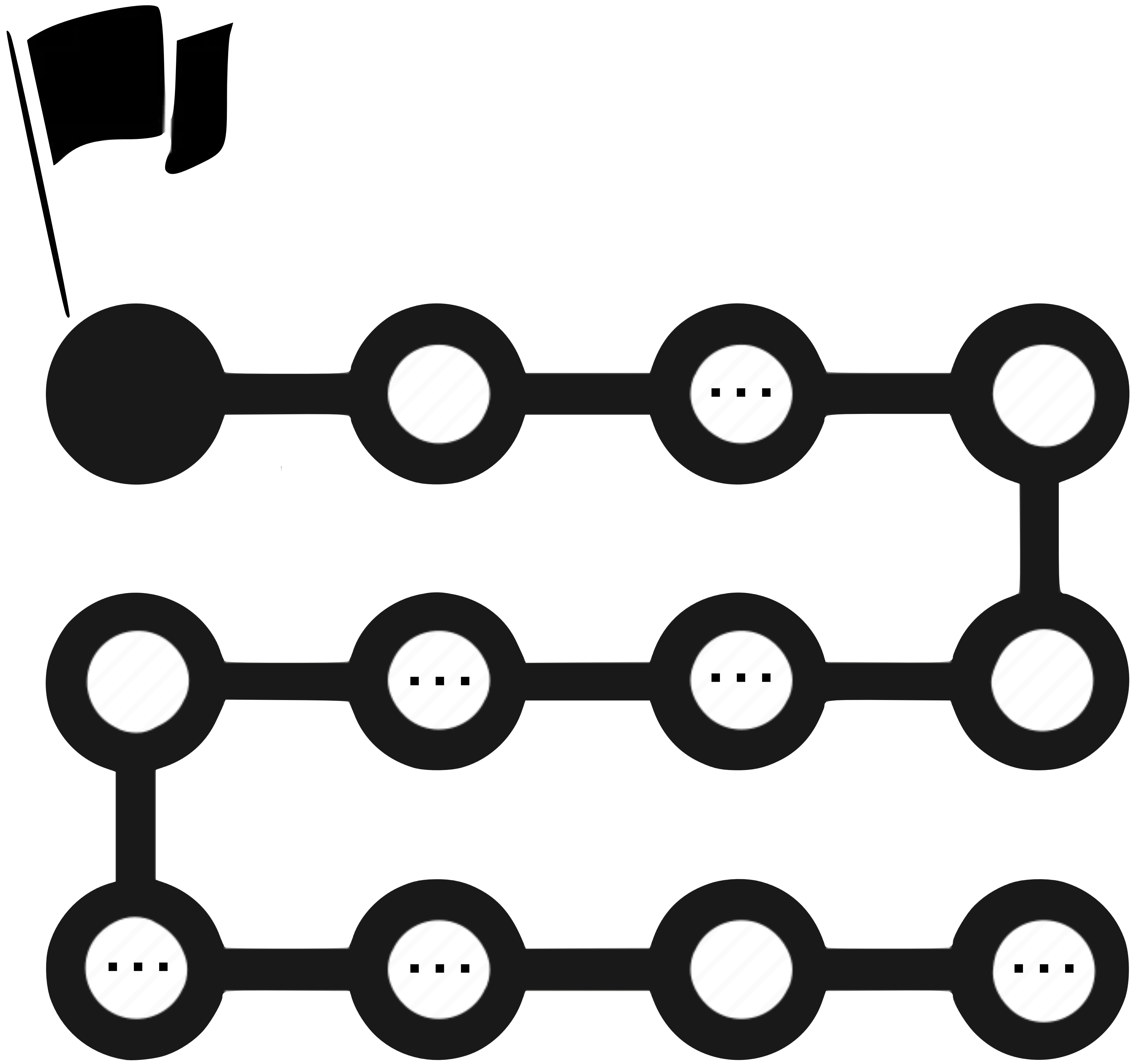
What is Secure Computation?



Example. n hospitals want to jointly perform statistical tests, or run ML algorithms, on the private data of their patients, to

- Uncover correlations between medical conditions and patient information
- Study the effect of medications
- Discover new treatments
- …

A Brief History of Secure Computation

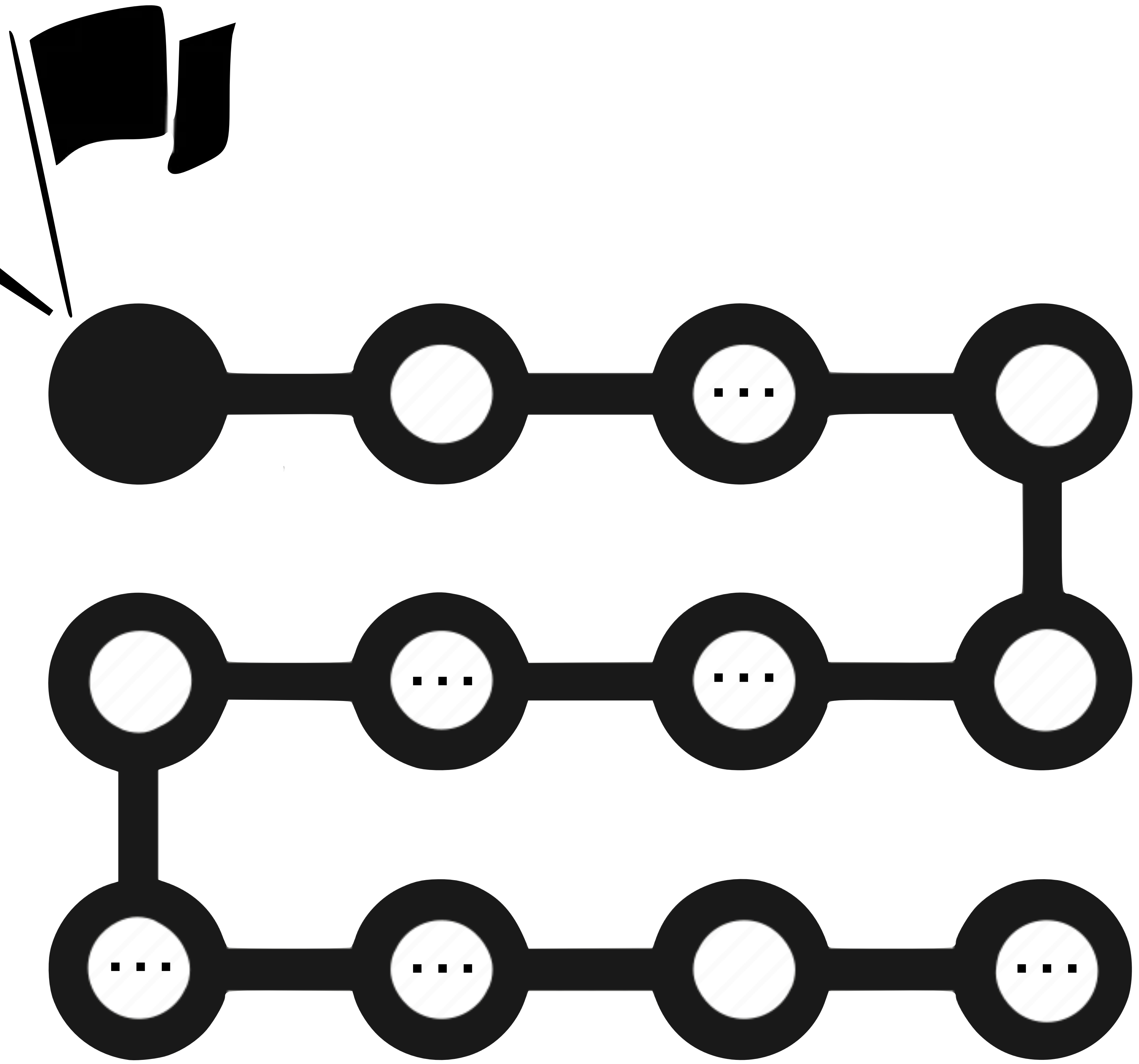


A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✔ Secure computation is possible in theory

✘ Very slow in practice: billions of expensive operations, TB of communication…



A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

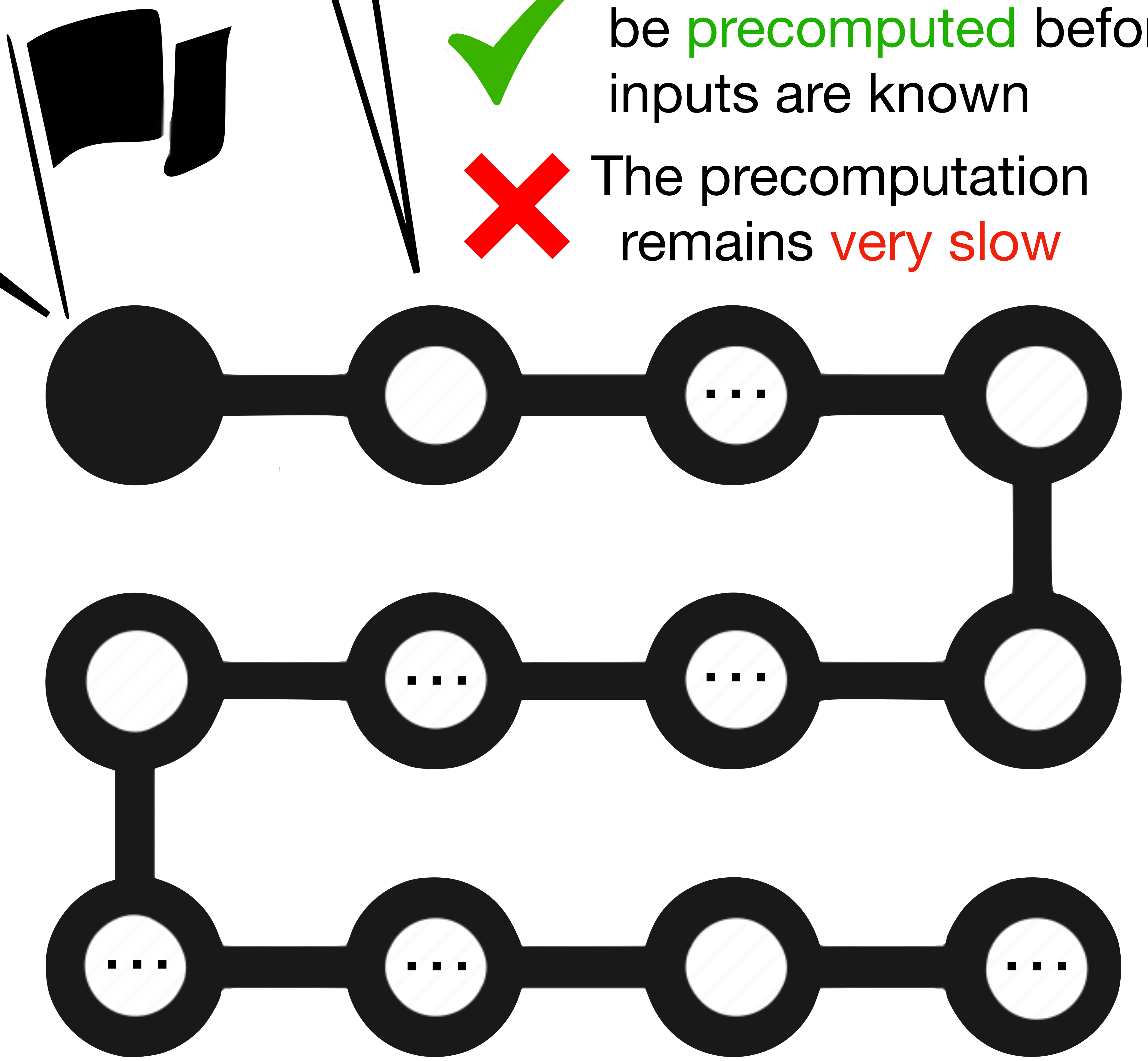
✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

✗ The precomputation remains very slow



A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

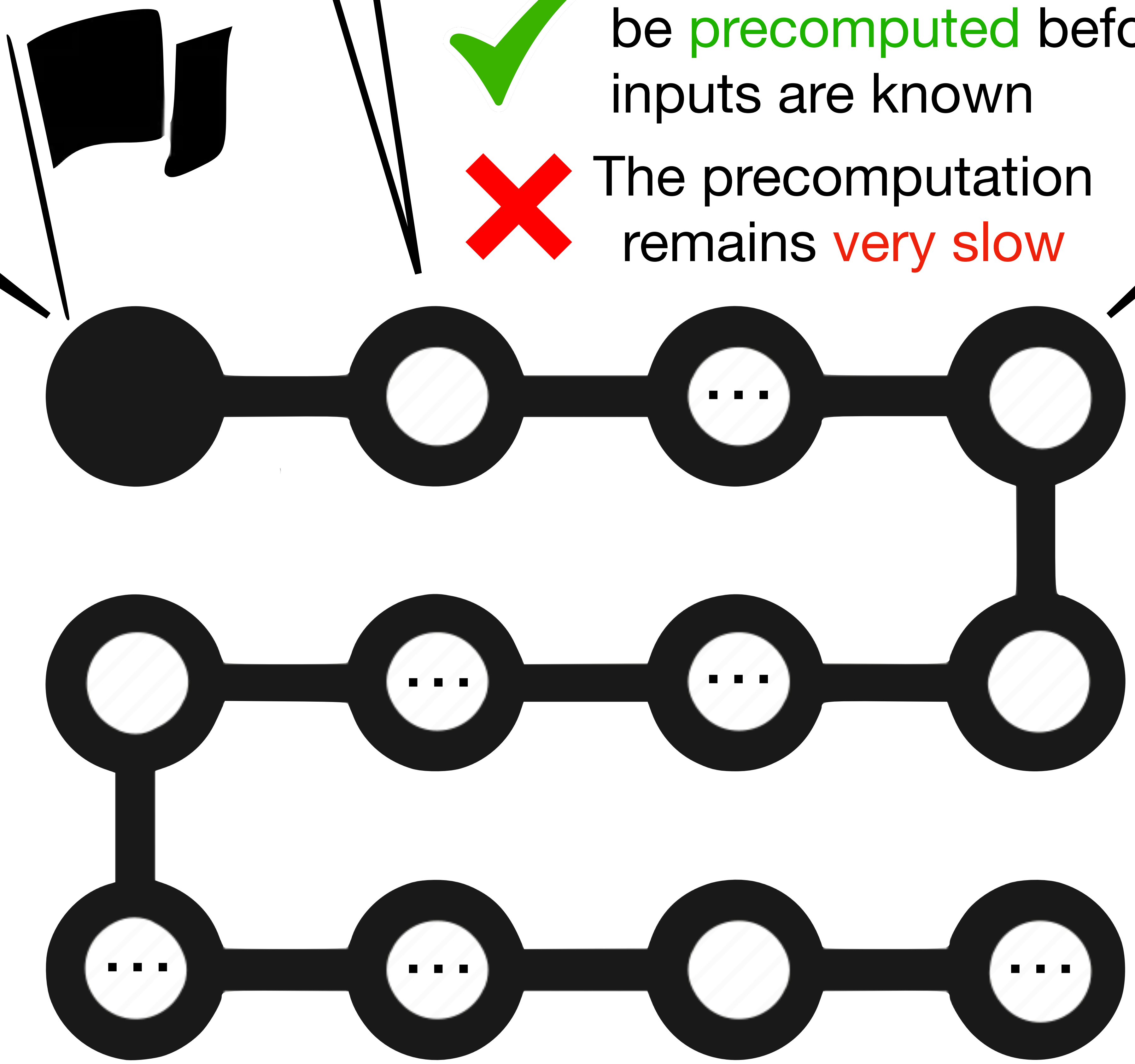
✗ The precomputation remains very slow

Beaver, 1996

OT extension

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication



A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

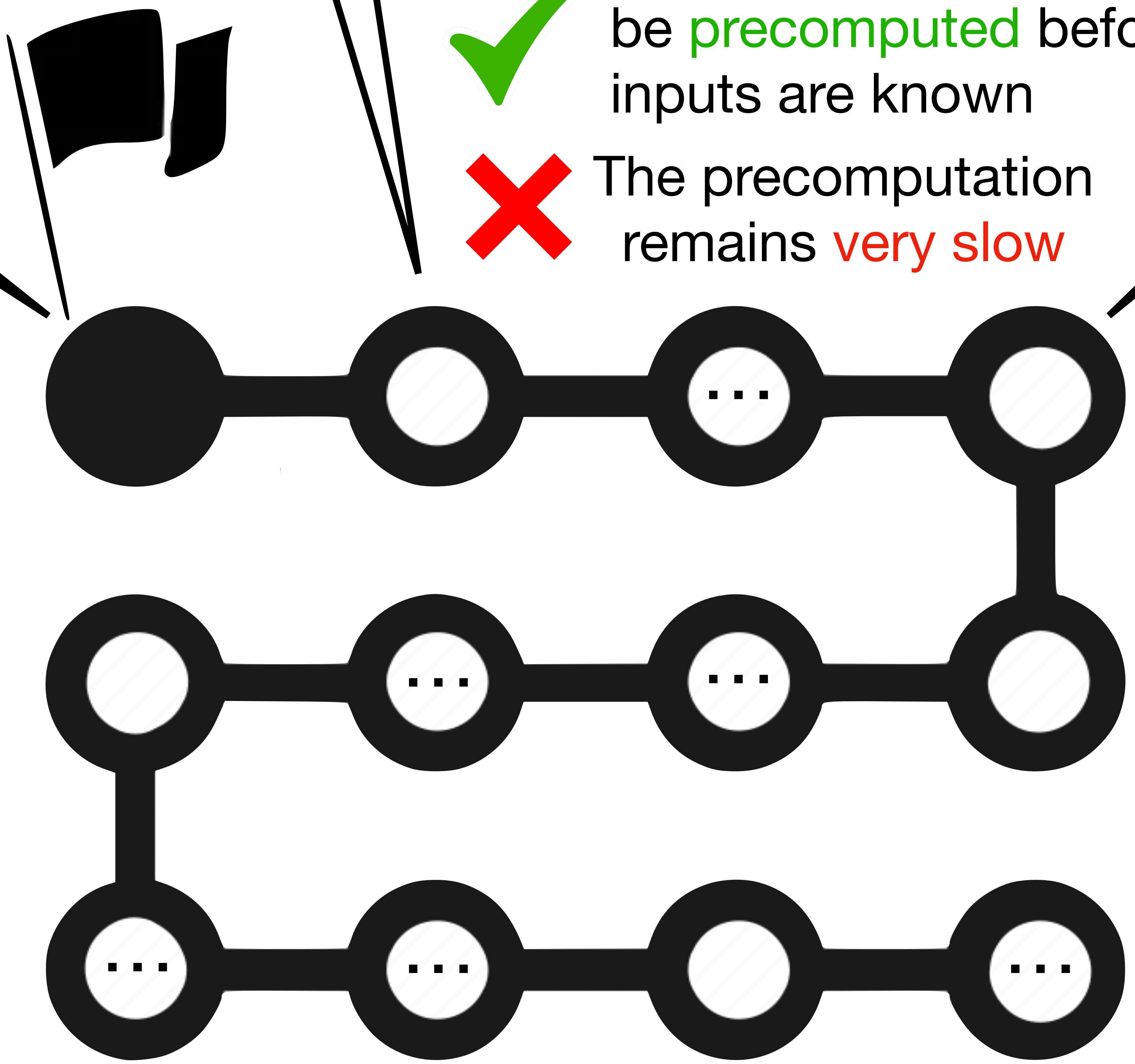
✗ The precomputation remains very slow

Beaver, 1996

OT extension

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication



IKNP, 2003

✓ Makes Beaver 1996 truly practical: MPC is now efficient!

✗ Still requires heavy communication

A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

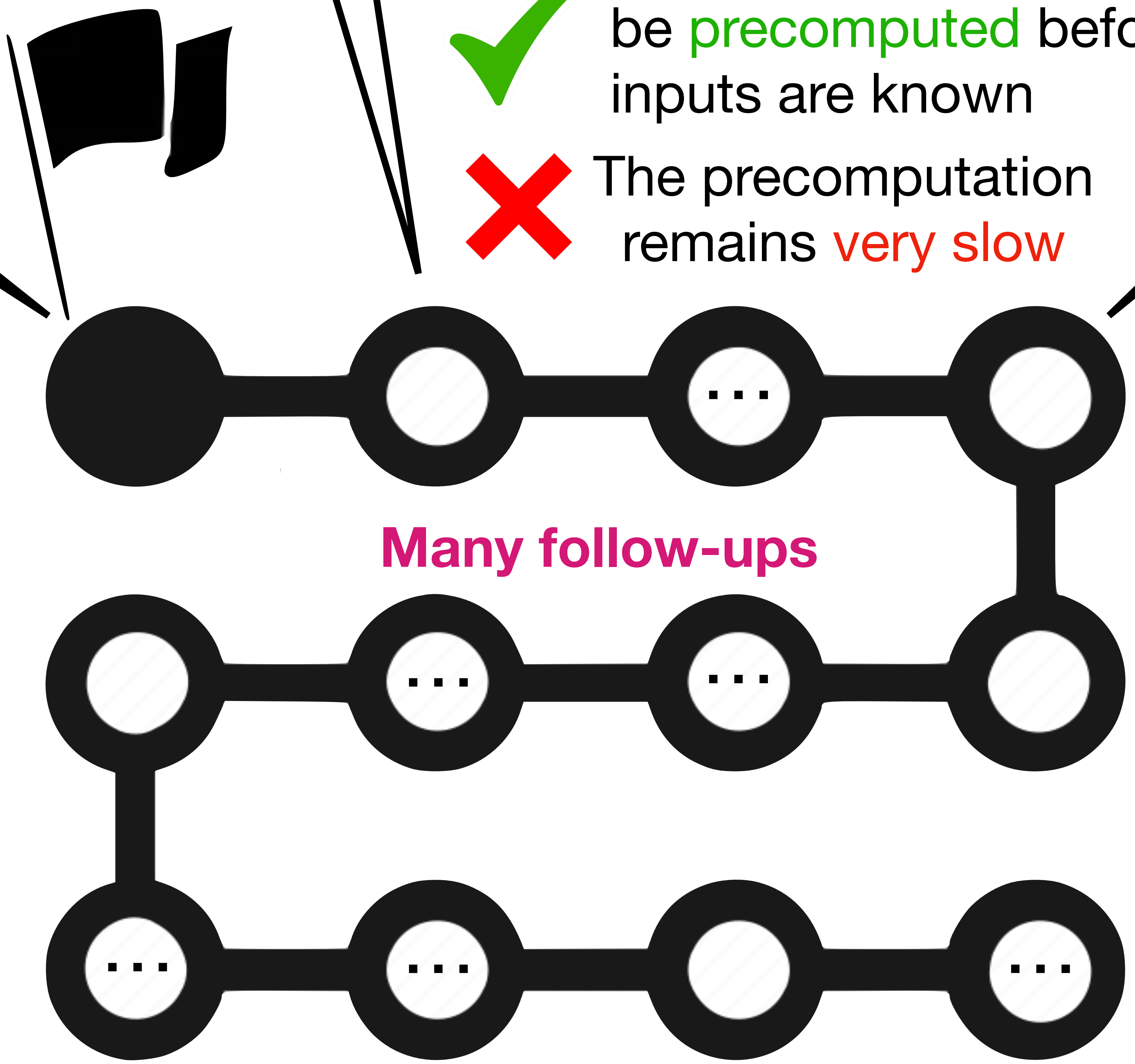
✗ The precomputation remains very slow

Beaver, 1996

OT extension

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication



IKNP, 2003

✓ Makes Beaver 1996 truly practical: MPC is now efficient!

✗ Still requires heavy communication

A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

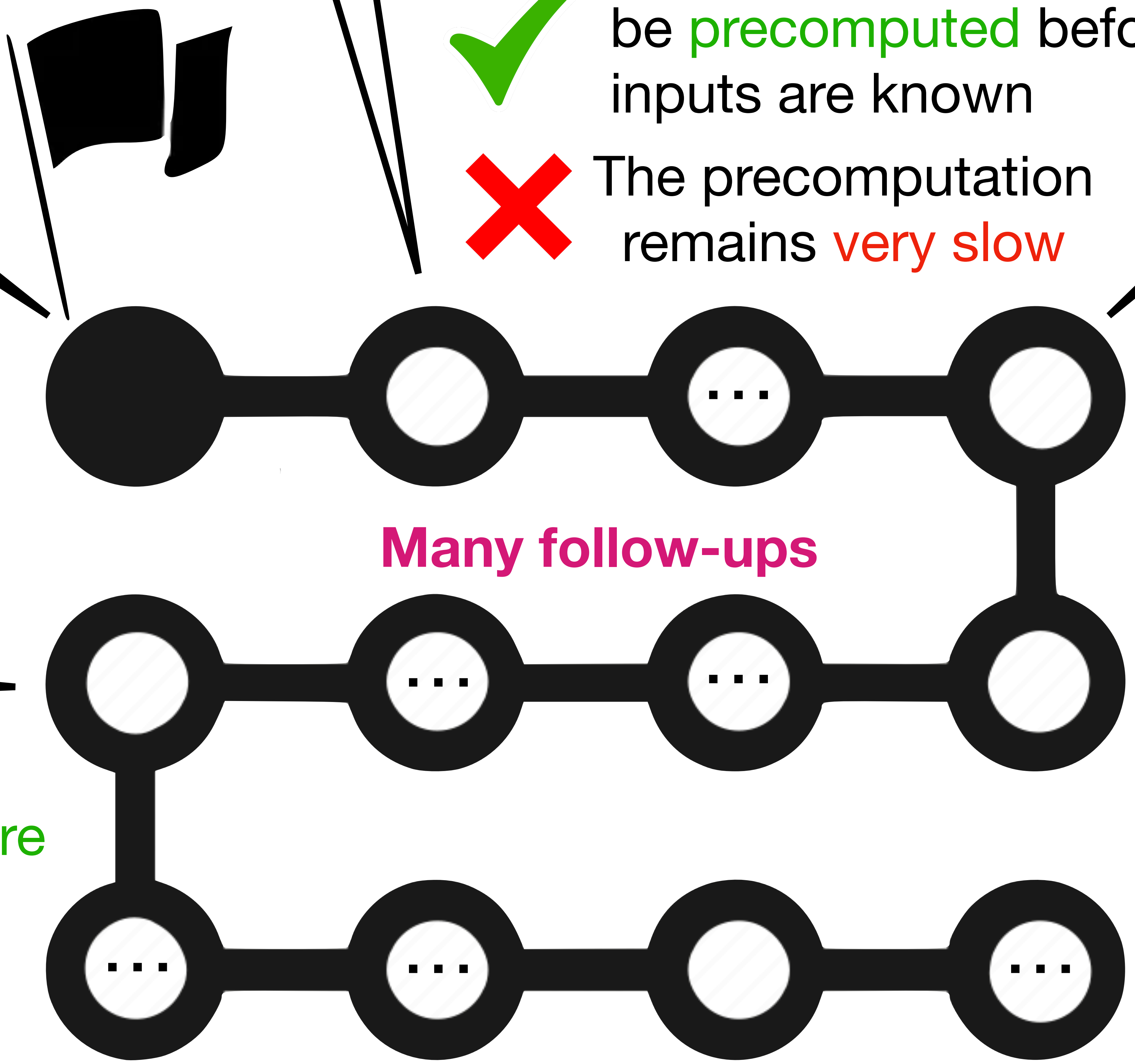
✗ The precomputation remains very slow

Beaver, 1996

OT extension

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication



IKNP, 2003

✓ Makes Beaver 1996 truly practical: MPC is now efficient!

✗ Still requires heavy communication

DPSZ, 2012

✓ Suddenly, n party, actively-secure MPC becomes a reality

✗ Still requires heavy comm.

The MPC explosion

Tons of follow-ups, improvements, first real-world deployments…

A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995

Correlated randomness

✓ Secure computation can be precomputed before inputs are known

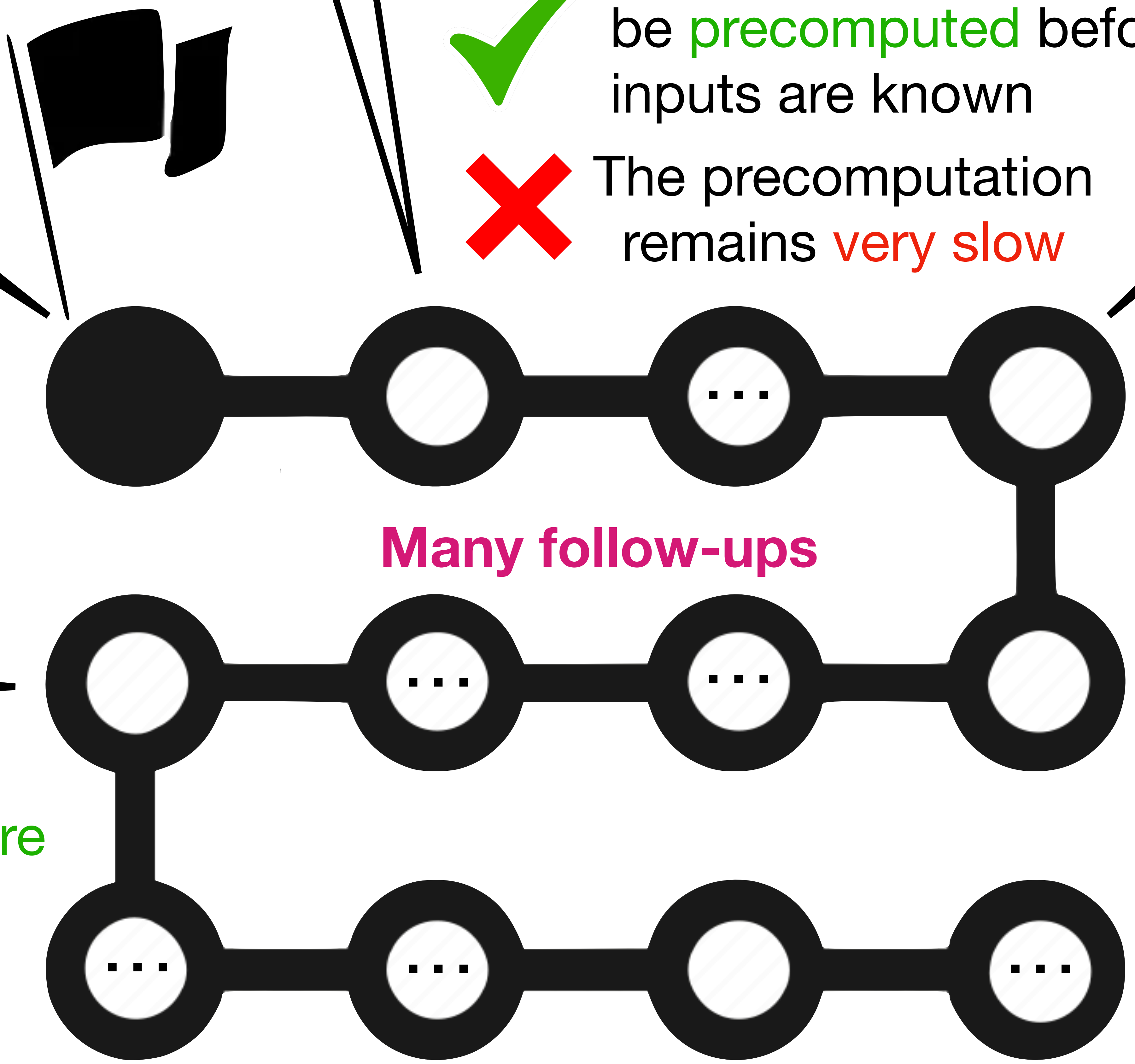
✗ The precomputation remains very slow

Beaver, 1996

OT extension

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication



Many follow-ups

IKNP, 2003

✓ Makes Beaver 1996 truly practical: MPC is now efficient!

✗ Still requires heavy communication

DPSZ, 2012

✓ Suddenly, n party, actively-secure MPC becomes a reality

✗ Still requires heavy comm.

Many follow-ups

The MPC explosion
Tons of follow-ups, improvements, first real-world deployments…

A Brief History of Secure Computation

Yao, 1986 (two parties)
GMW, 1987 (n parties)

✓ Secure computation is possible in theory

✗ Very slow in practice: billions of expensive operations, TB of communication…

Beaver, 1995 — **Correlated randomness**

✓ Secure computation can be precomputed before inputs are known

✗ The precomputation remains very slow

Beaver, 1996 — **OT extension**

✓ Almost all expensive operations can be replaced by cheap operations

✗ Mostly theoretical result, and still requires heavy communication

Many follow-ups

IKNP, 2003

✓ Makes Beaver 1996 truly practical: MPC is now efficient!

✗ Still requires heavy communication

DPSZ, 2012

✓ Suddenly, n party, actively-secure MPC becomes a reality

✗ Still requires heavy comm.

The MPC explosion
Tons of follow-ups, improvements, first real-world deployments…

Many follow-ups

BCGI, 2018 — **Pseudorandom correlations**

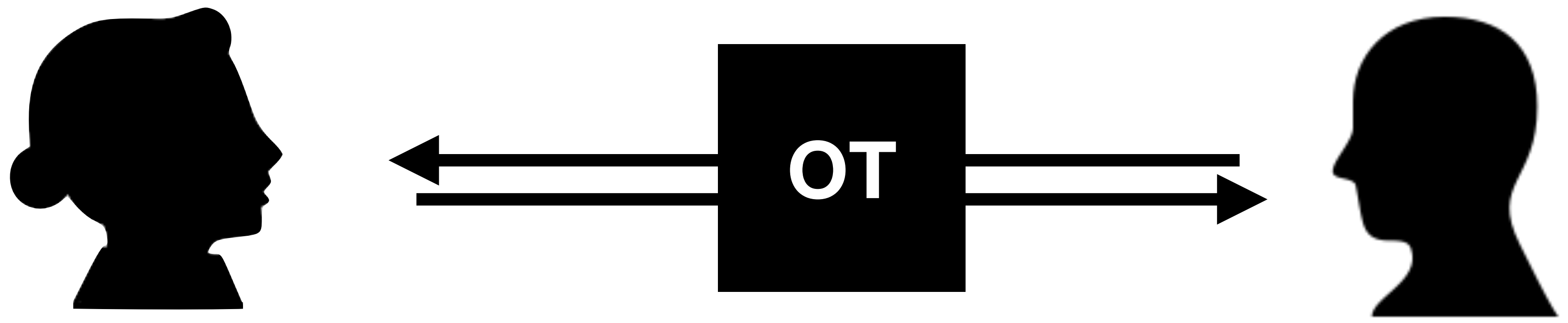
✓ Only lightweight communication

✗ Mostly for two parties

Secure Computation from Oblivious Transfer

Oblivious Transfer

A minimal example of secure computation…



(s_0, s_1) b

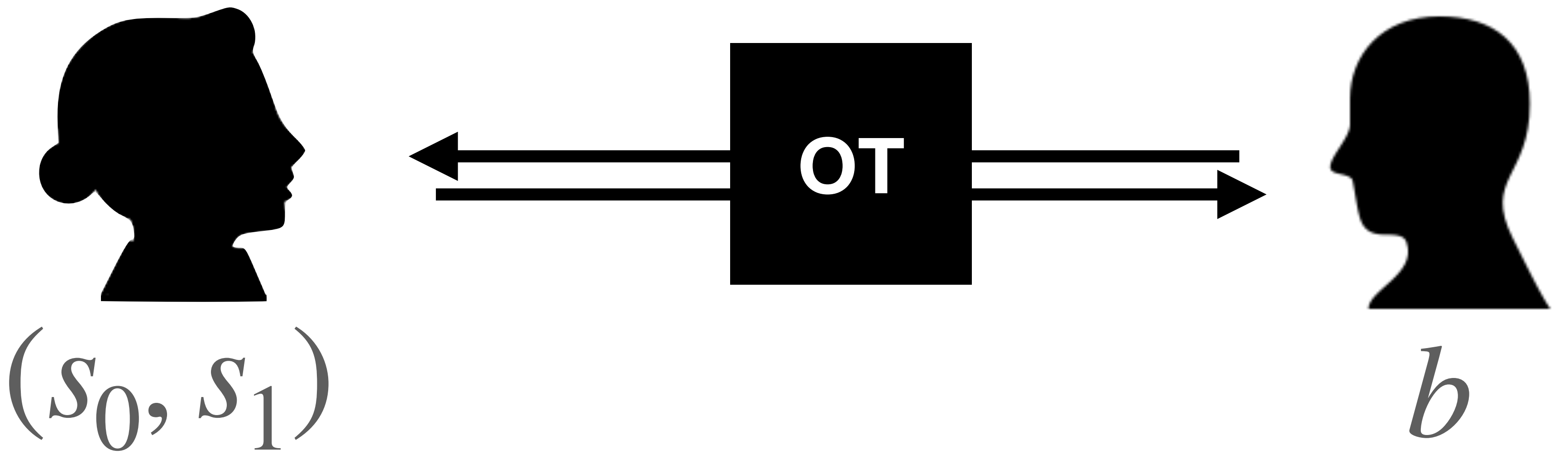
Output: Bob learns s_b

Security: Alice does not learn b, Bob does not learn s_{1-b}.

Secure Computation from Oblivious Transfer

Oblivious Transfer

A minimal example of secure computation…



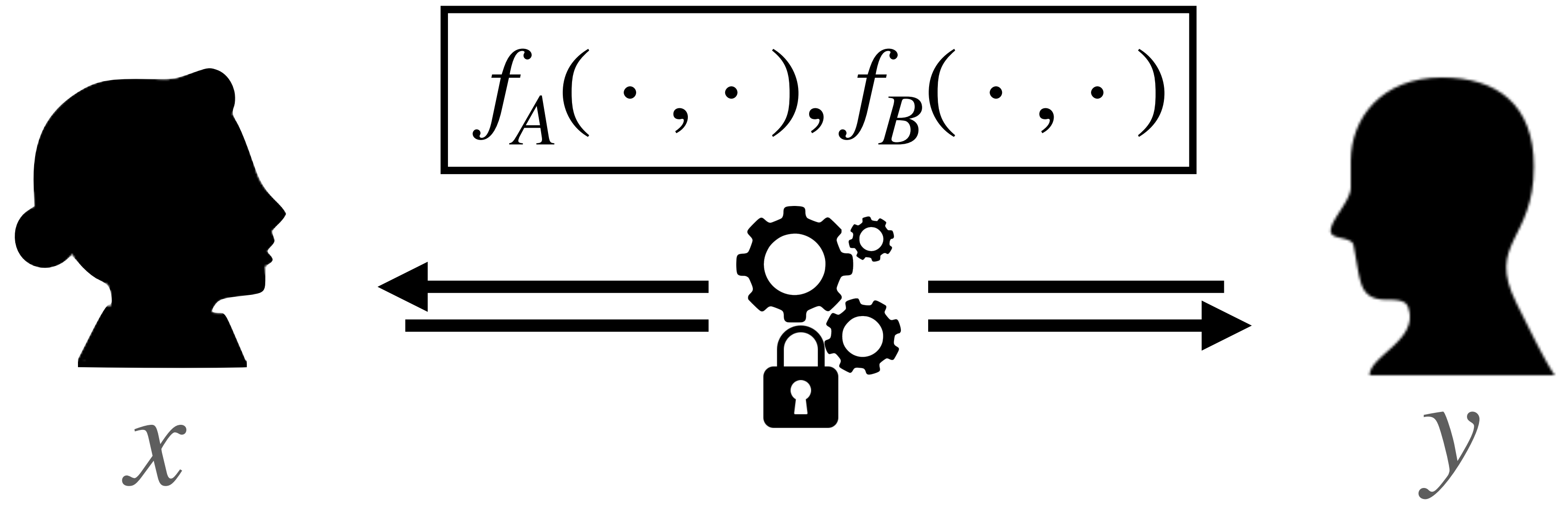
Output: Bob learns s_b

Security: Alice does not learn b, Bob does not learn s_{1-b}.

GMW, 1987

Secure Computation for all functions

Which suffices for all functions!



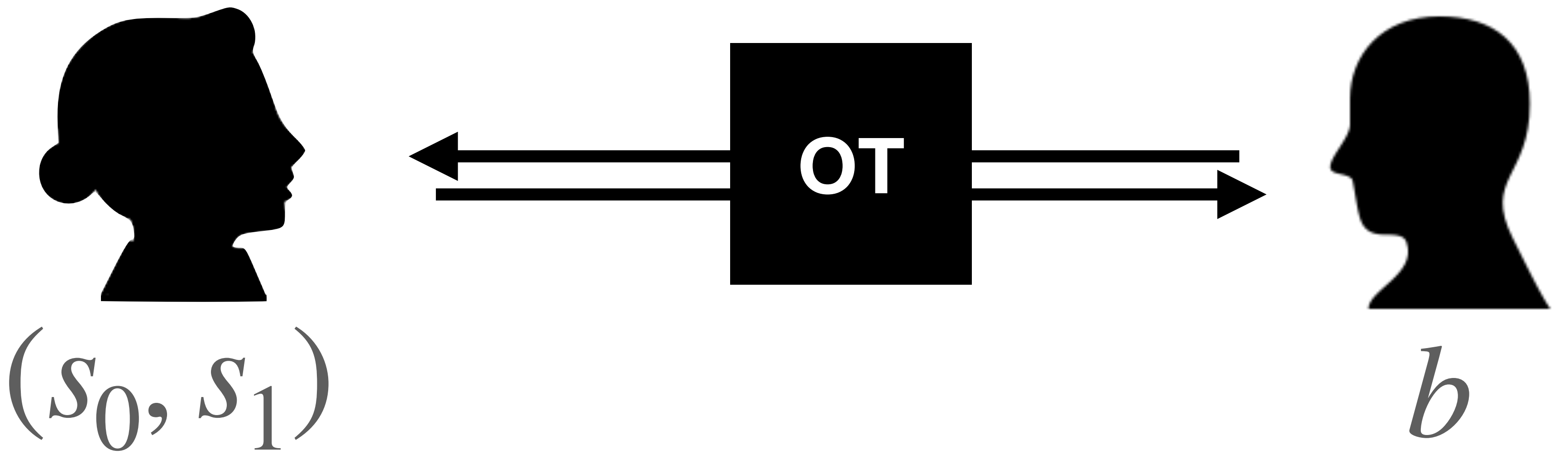
Output: Alice learns $f_A(x, y)$, Bob learns $f_B(x, y)$

Security: Alice and Bob learn nothing else

Secure Computation from Oblivious Transfer

Oblivious Transfer

A minimal example of secure computation…



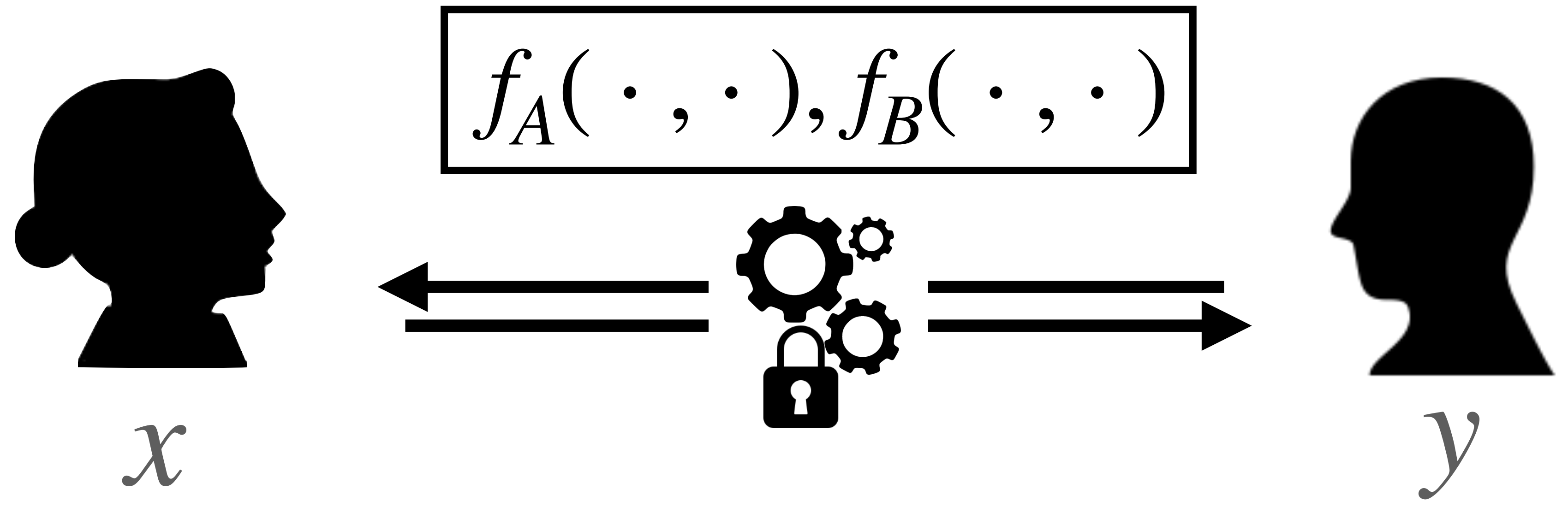
Output: Bob learns s_b

Security: Alice does not learn b, Bob does not learn s_{1-b}.

GMW, 1987

Secure Computation for all functions

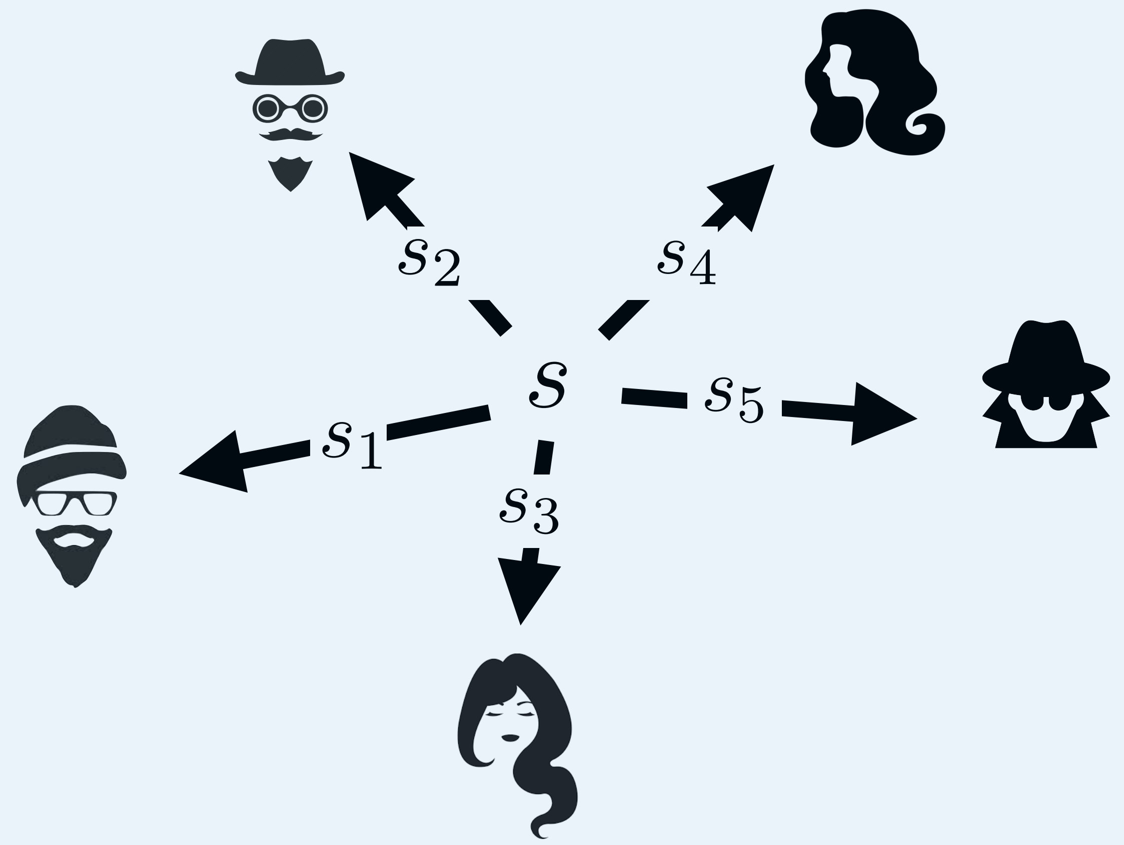
Which suffices for all functions!



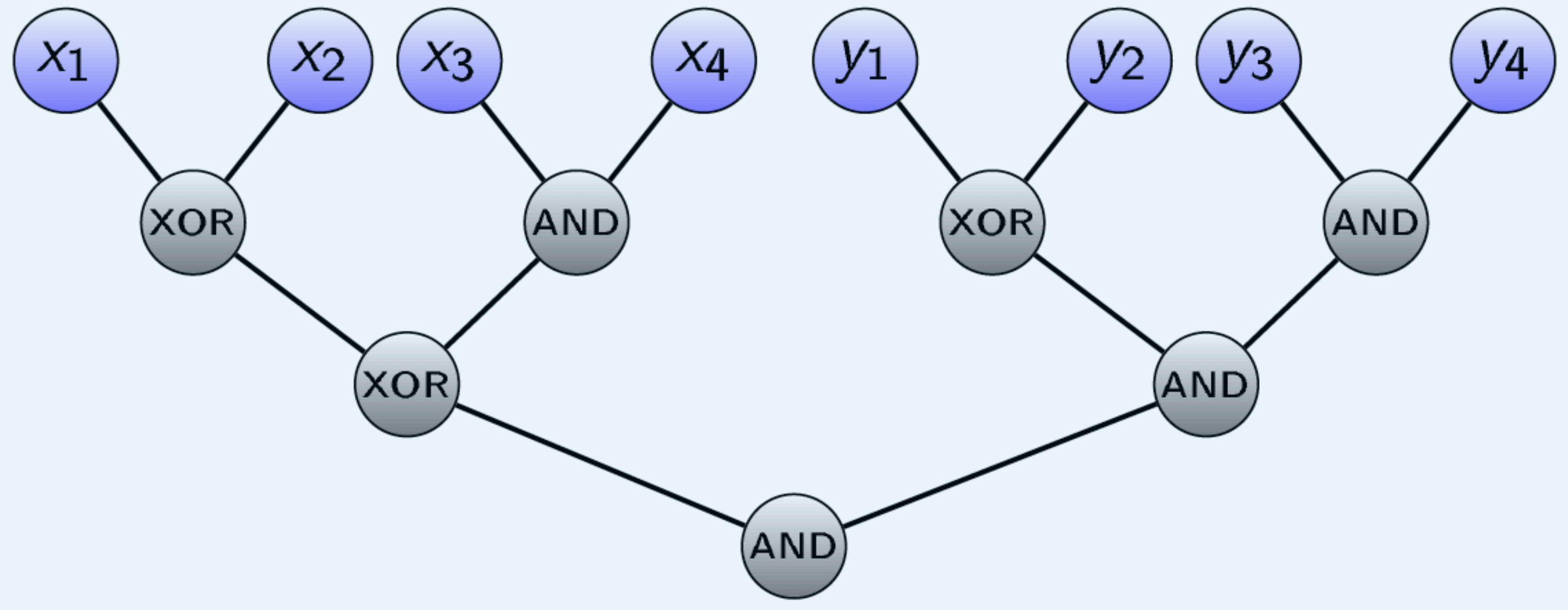
Output: Alice learns $f_A(x, y)$, Bob learns $f_B(x, y)$

Security: Alice and Bob learn nothing else

1. Use (additive) secret sharing



2. Write the function as a circuit



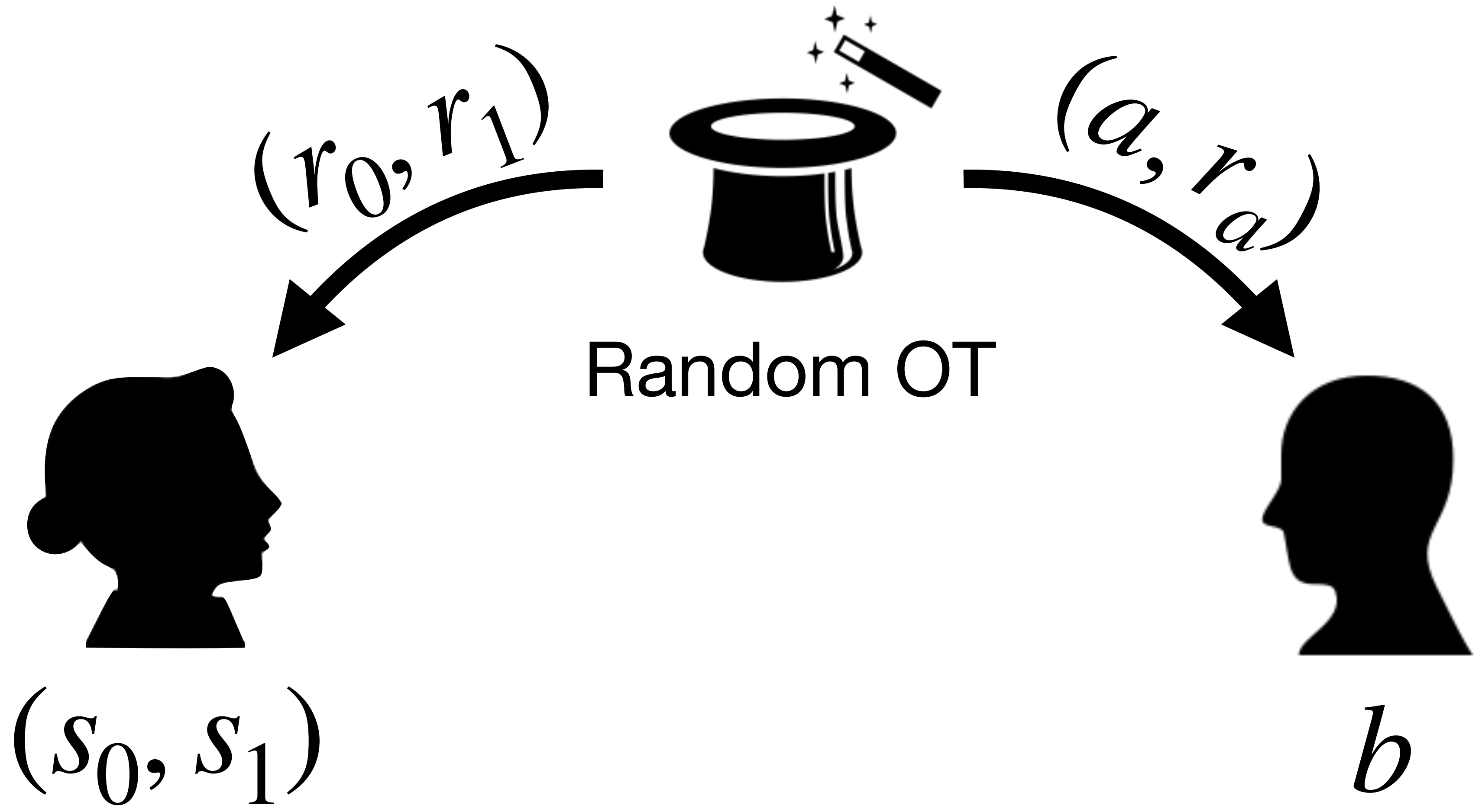
3. Use OT to compute the gates

share(x, y) $\Longrightarrow$ share(GATE(x, y))

I'll skip the details for now, but feel free to ask for them!

Precomputing Oblivious Transfers (Beaver, 1995)

Given a **random** oblivious transfer, two parties can construct a **standard** oblivious transfer

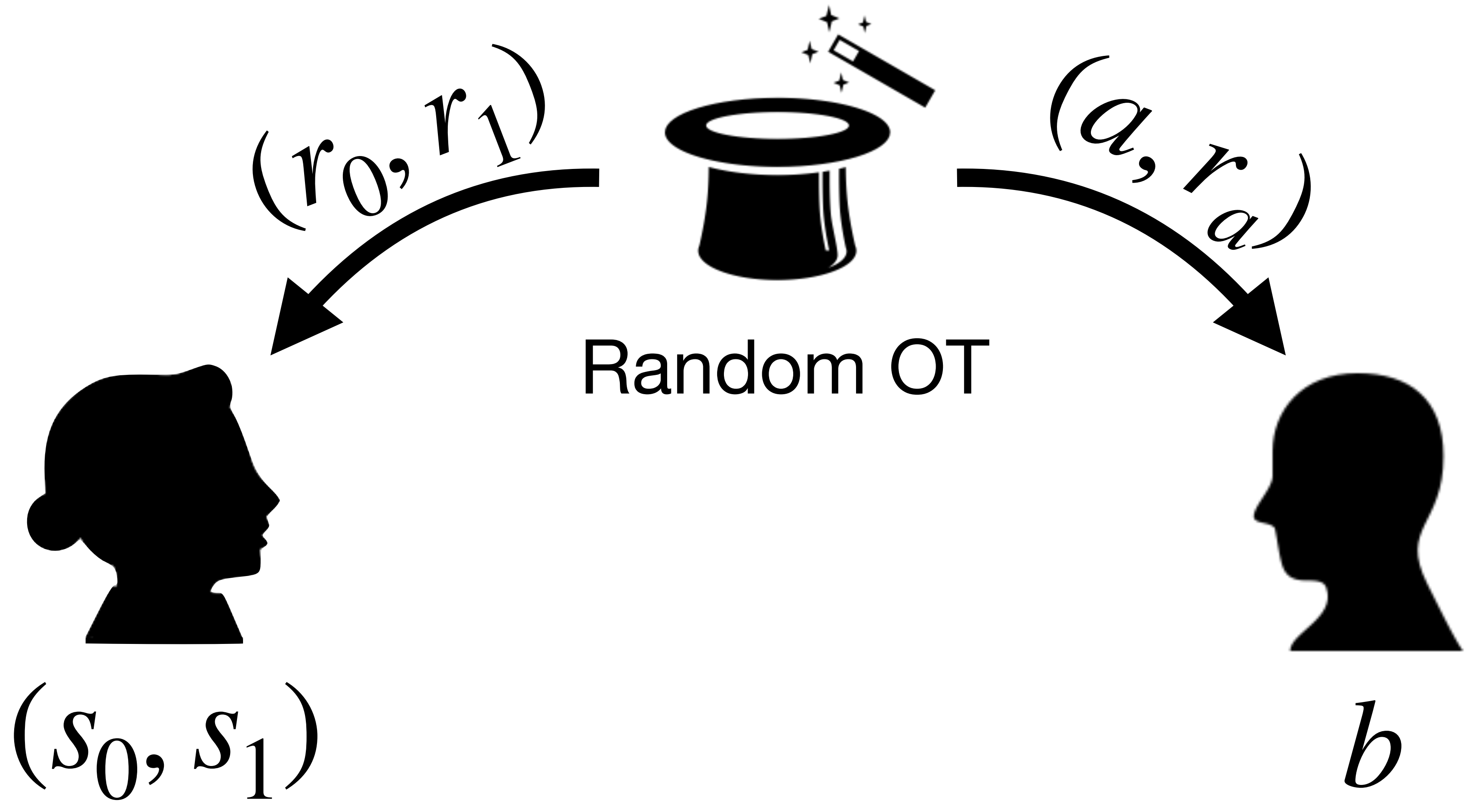


The (simple) protocol:

- If $a = b$ and Bob gets $(s_0 \oplus r_0, s_1 \oplus r_1)$, he can get $s_b = s_a$, since he knows only $r_b = r_a$.
- If $a = 1 - b$ and Bob gets $(s_0 \oplus r_1, s_1 \oplus r_0)$, he again gets s_b, since he knows only r_{1-b}.
- Bob simply tells Alice whether $a = b$ (leaks nothing since a is random!), and Alice sends the appropriate pair.

Precomputing Oblivious Transfers (Beaver, 1995)

Given a **random** oblivious transfer, two parties can construct a **standard** oblivious transfer



The (simple) protocol:

- If $a = b$ and Bob gets $(s_0 \oplus r_0, s_1 \oplus r_1)$, he can get $s_b = s_a$, since he knows only $r_b = r_a$.
- If $a = 1 - b$ and Bob gets $(s_0 \oplus r_1, s_1 \oplus r_0)$, he again gets s_b, since he knows only r_{1-b}.
- Bob simply tells Alice whether $a = b$ (leaks nothing since a is random!), and Alice sends the appropriate pair.

The protocol is:

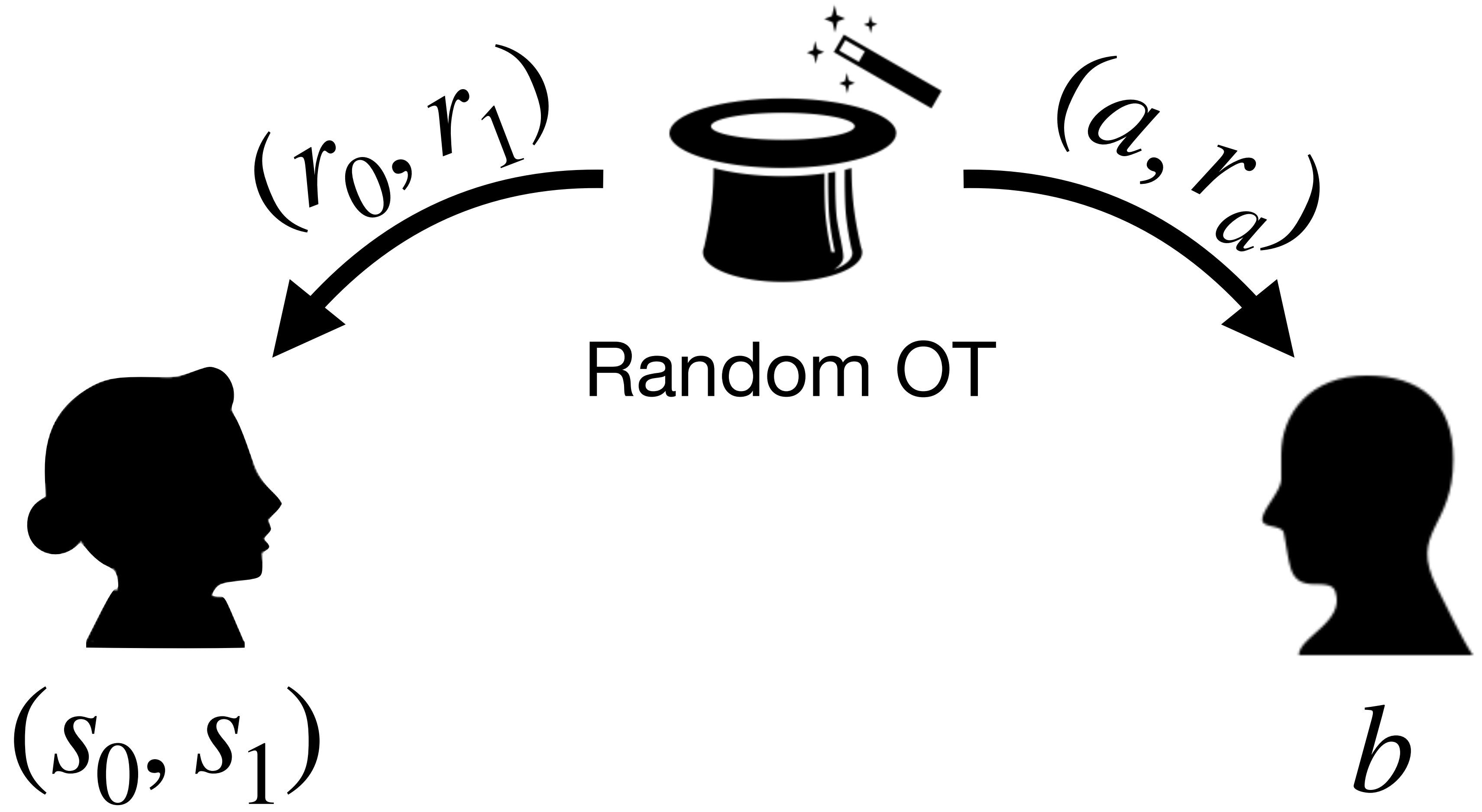
- Perfectly secure (no assumption required)
- Very fast: only three bits exchanged per OT

$\Longrightarrow$ Almost all computations can be executed **ahead of time** to precompute many OTs

$\Longrightarrow$ Reduces *efficient secure computation* to the task of securely and efficiently **distributing long correlated strings** (here, random pairs (r_0, r_1) an (a, r_a))

Precomputing Oblivious Transfers (Beaver, 1995)

Given a **random** oblivious transfer, two parties can construct a **standard** oblivious transfer



The protocol is:

- Perfectly secure (no assumption required)
- Very fast: only three bits exchanged per OT

$\Longrightarrow$ Almost all computations can be executed **ahead of time** to precompute many OTs

$\Longrightarrow$ Reduces *efficient secure computation* to the task of securely and efficiently **distributing long correlated strings** (here, random pairs (r_0, r_1) an (a, r_a))

The (simple) protocol:

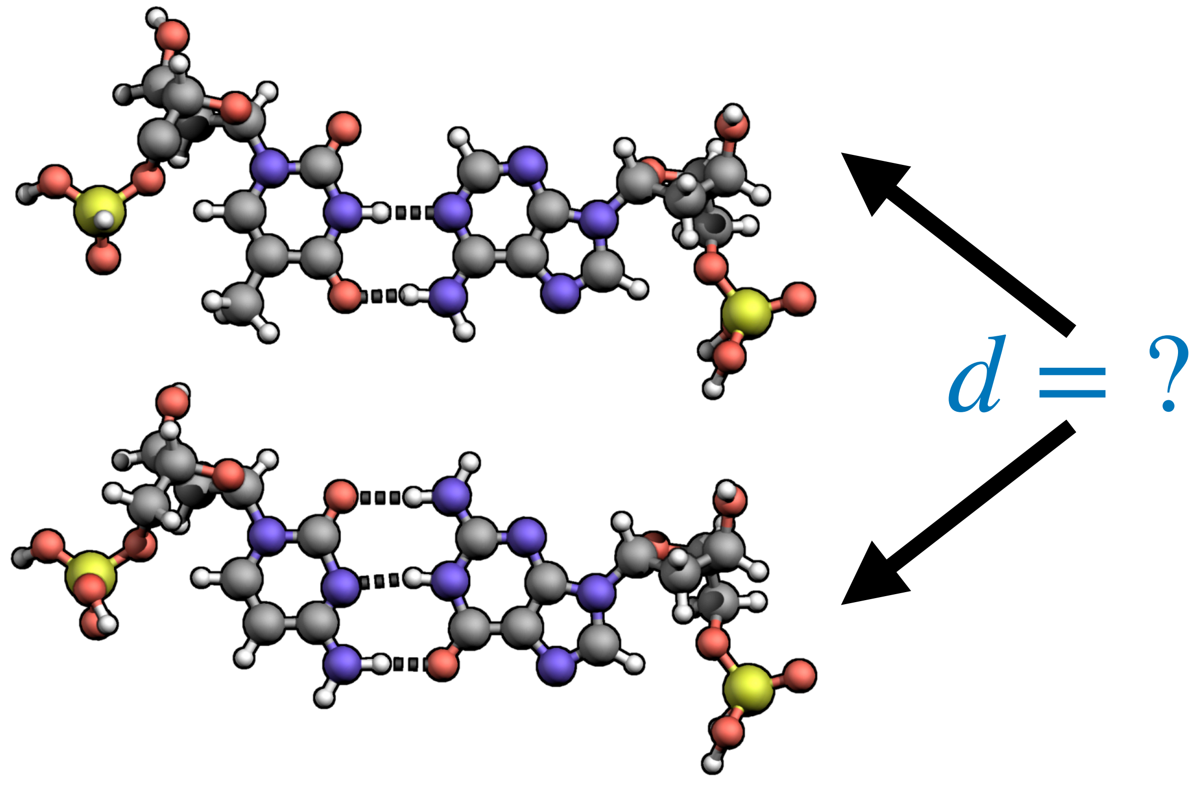
- If $a = b$ and Bob gets $(s_0 \oplus r_0, s_1 \oplus r_1)$, he can get $s_b = s_a$, since he knows only $r_b = r_a$.
- If $a = 1 - b$ and Bob gets $(s_0 \oplus r_1, s_1 \oplus r_0)$, he again gets s_b, since he knows only r_{1-b}.
- Bob simply tells Alice whether $a = b$ (leaks nothing since a is random!), and Alice sends the appropriate pair.

Ishai-Killian-Nissim-Petrank 2003:

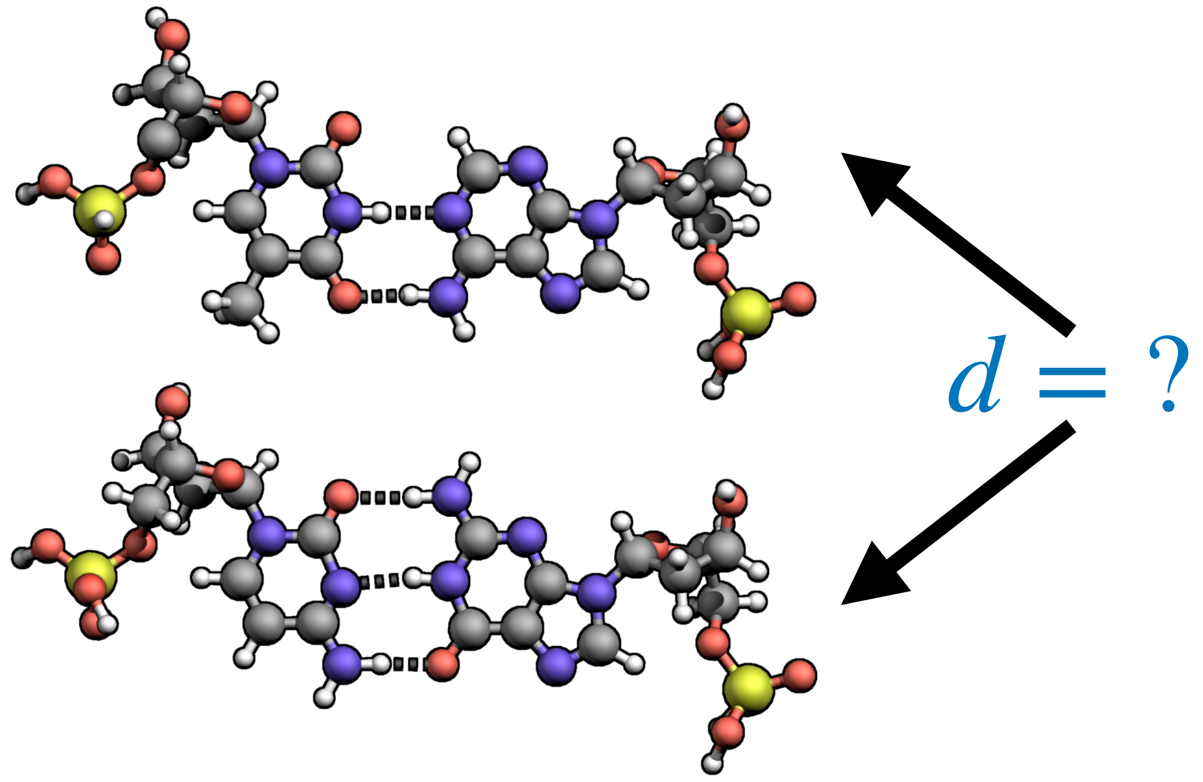
Computing n random OTs can be done using
✔ 128 « base » oblivious transfers
✔ 3 **evaluations of a hash function** per OT
✘ $\sim$ 100 bits of communication per OT

Just to Get a Sense of Scales…

- **Edit distance:** number of insertions, deletions, and substitutions to convert one string into another
- Widely used to measure similarities, e.g. in genomics
- This is by all mean a relatively **simple function**



Just to Get a Sense of Scales…



- **Edit distance:** number of insertions, deletions, and substitutions to convert one string into another
- Widely used to measure similarities, e.g. in genomics
- This is by all mean a relatively **simple function**

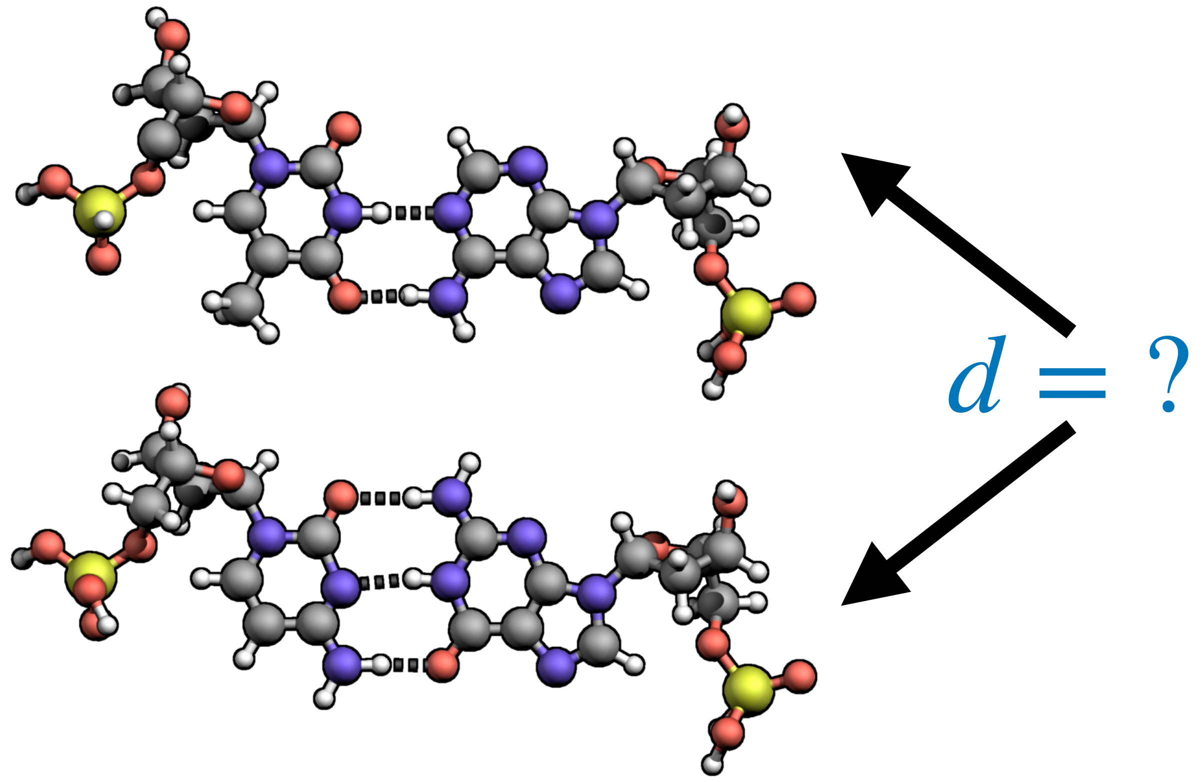
Assume Alice and Bob want to securely compute the edit distance between 512-byte inputs (that is, *small* inputs). This requires:

- Converting the function to a boolean circuit $\Longrightarrow$ 5,901,194,475 AND gates according to [1]
- Securely computing the circuit $\Longrightarrow$ 5,901,194,475 $\times$ 100 bits $\approx$ 70 Gigabytes of communication

This is **doable but expensive**, and communication is **typically the bottleneck** in secure computation protocols.

[1] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Proceedings of the 21st USENIX conference on Security symposium, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

Just to Get a Sense of Scales…



- **Edit distance:** number of insertions, deletions, and substitutions to convert one string into another
- Widely used to measure similarities, e.g. in genomics
- This is by all mean a relatively **simple function**

Assume Alice and Bob want to securely compute the edit distance between 512-byte inputs (that is, *small* inputs). This requires:

- Converting the function to a boolean circuit $\Longrightarrow$ 5,901,194,475 AND gates according to [1]
- Securely computing the circuit $\Longrightarrow$ 5,901,194,475 $\times$ 100 bits $\approx$ 70 Gigabytes of communication

This is **doable but expensive**, and communication is **typically the bottleneck** in secure computation protocols.

$\Longrightarrow$ **Can we precompute random OTs using much less communication?**

[1] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Proceedings of the 21st USENIX conference on Security symposium, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

Back to Secure Computation

Pseudorandom correlation generators, introduced in my CCS'2018 paper with Boyle, Gilboa, and Ishai, provide a way to generate n *pseudo*-random OTs using **almost no communication**

Back to Secure Computation

Pseudorandom correlation generators, introduced in my CCS'2018 paper with Boyle, Gilboa, and Ishai, provide a way to generate n *pseudo*-random OTs using **almost no communication**

Ishai-Killian-Nissim-Petrank 2003:

Computing n random OTs can be done using
- ✓ 128 « base » oblivious transfers
- ✓ 3 **evaluations of a hash function** per OT
- ✗ ~ 100 bits of communication per OT

Boyle-C-Gilboa-Ishai 2018 and **Boyle-C-Gilboa-Ishai-Kohl-Scholl 2019**:

Computing n random OTs can be done using
- ✓ A few hundred « base » oblivious transfers
- ✓ 2 **evaluations of a hash function** per OT
- ✓ **~ 0 bits of communication per OT**
- ? Computing an n-by-$2n$ matrix-vector product

Back to Secure Computation

Pseudorandom correlation generators, introduced in my CCS'2018 paper with Boyle, Gilboa, and Ishai, provide a way to generate n *pseudo*-random OTs using **almost no communication**

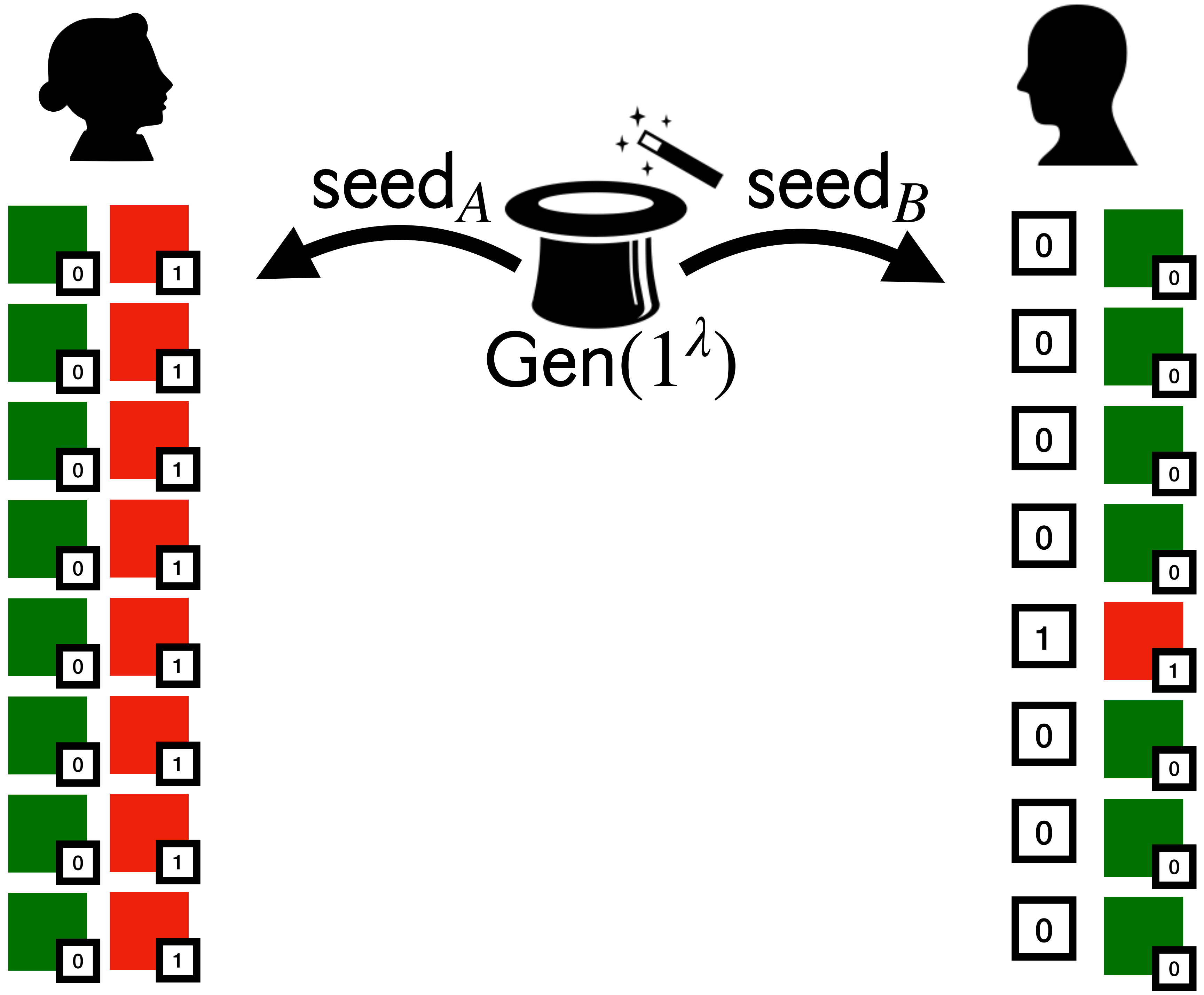
Ishai-Killian-Nissim-Petrank 2003:	**Boyle-C-Gilboa-Ishai 2018** and **Boyle-C-Gilboa-Ishai-Kohl-Scholl 2019**:
Computing n random OTs can be done using ✔ 128 « base » oblivious transfers ✔ 3 **evaluations of a hash function** per OT ✘ $\sim$ 100 bits of communication per OT	Computing n random OTs can be done using ✔ A few hundred « base » oblivious transfers ✔ 2 **evaluations of a hash function** per OT ✔ $\sim$ **0 bits of communication per OT** ? Computing an n-by-$2n$ matrix-vector product

- Choosing the « right » matrix is related to deep questions in coding theory
- Latest exciting works (**C**RR'21, B**C**GIKS'22) provide **extremely efficient instantiations**
- Many fundamental questions remain partially open:
  - ➡ Achieving more powerful correlations (related to deep questions in algebraic coding theory)
  - ➡ Extending efficiently to n parties (currently works best for two parties)
  - ➡ …

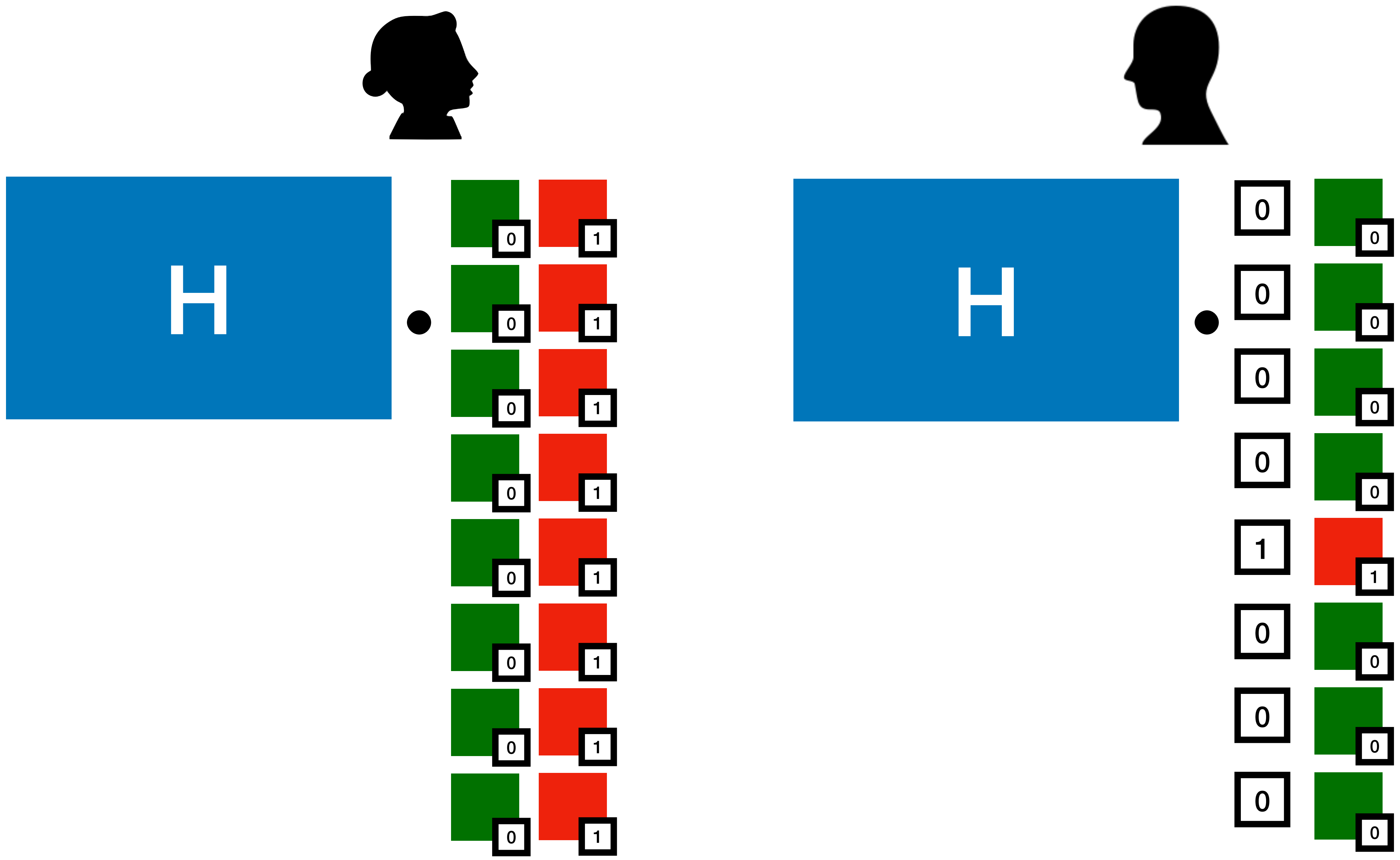
A 10s Walkthrough of the Core Ideas

Reminder: Alice and Bob want to get many (pseudorandom) oblivious transfers from *short* seeds.

Step 1. Design a strategy, using cryptographic techniques, to get a solution when Bob's selection bits are **all equal to 0** except t.



Step 2. **Scramble** the bits using a large, public, **structured**, compressive matrix multiplication



The natural way to attack is to distinguish from random by looking for a *bias* in $H \cdot \vec{b}$, i.e., finding $\vec{v}$ s.t. $\vec{v}^\intercal \cdot H \cdot \vec{b}$ is **biased**

$\iff \langle \vec{v} \cdot H, \vec{b} \rangle = 0$ with high probability

$\iff \vec{v}$ has low weight… Which is impossible when $H^\intercal$ generates a **good code**

$\implies$ **the goal is to find structured good codes where the computation of $x \to H^\intercal \cdot x$ is very fast**

Thank You for Your Attention!

Questions?

Licenses

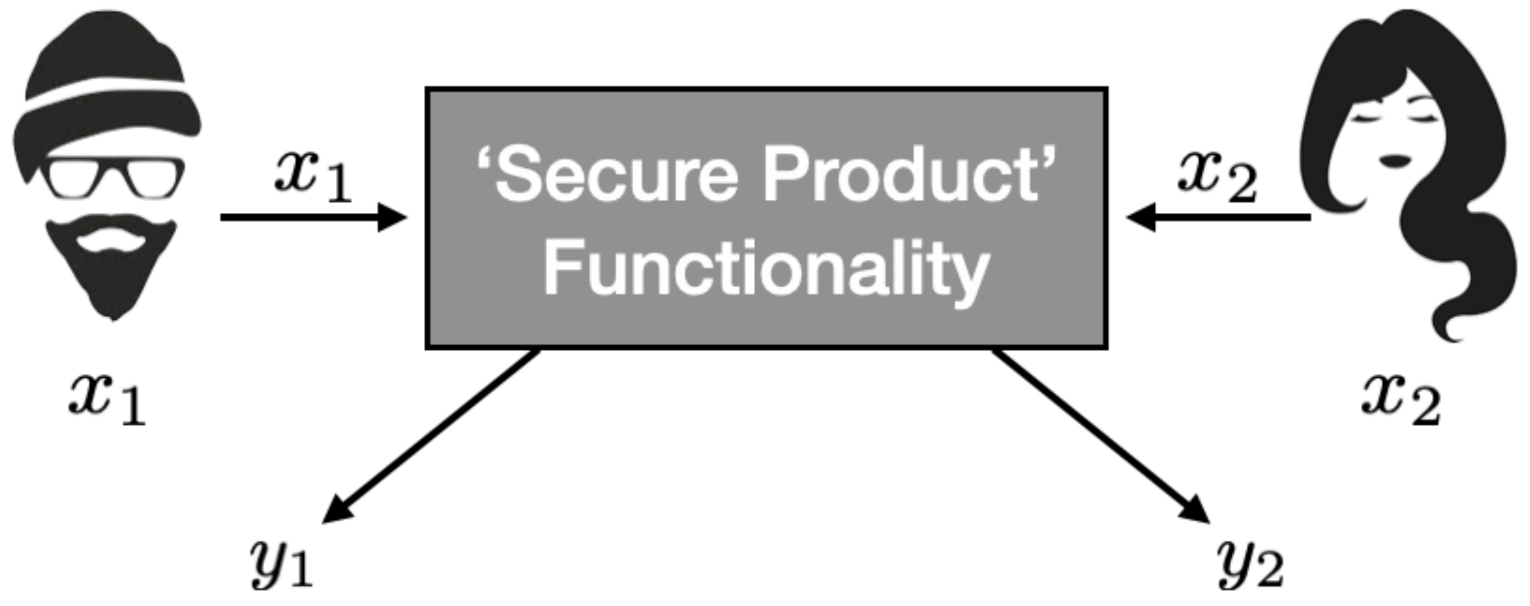
All images used in this talk are either

- Made on Keynote directly, or
- Taken from Wikimedia common, under the ShareAlike 3.0 (CC BY-SA 3.0) license (https://creativecommons.org/licenses/by-sa/3.0/), or
- Taken from Pixabay, under the public domain certification (https://creativecommons.org/licenses/publicdomain/)

Backup Slides

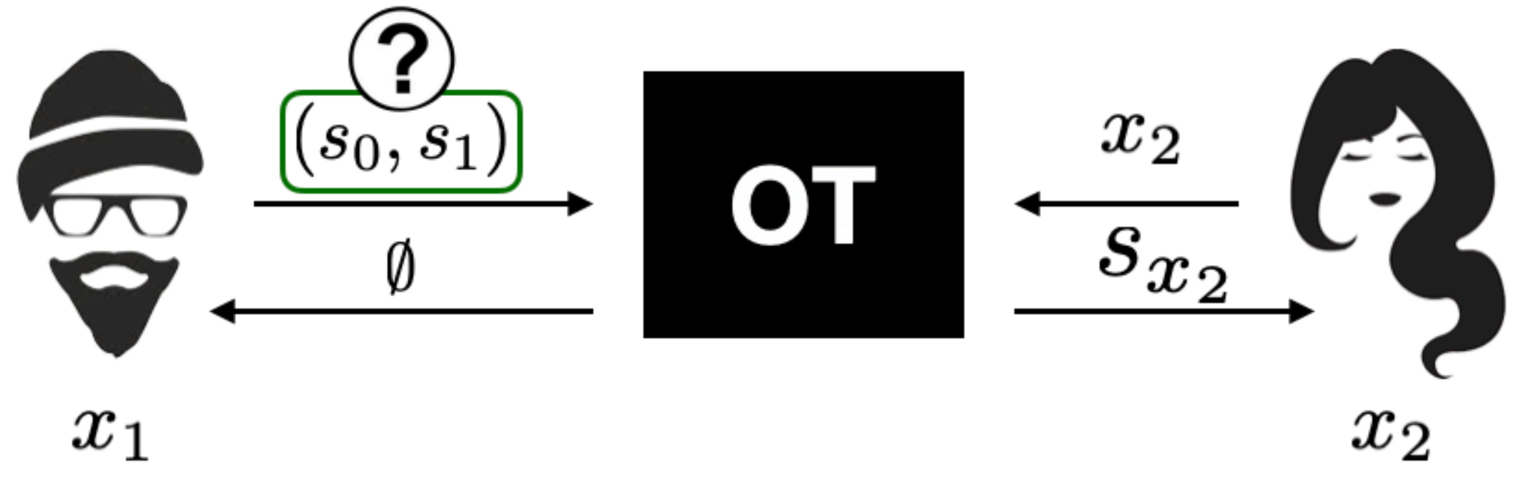
Secure Computation from Oblivious Transfer

Warm-up I: 2-Party Product Sharing



(y_1, y_2) random conditioned on $y_1 \oplus y_2 = x_1 x_2$

Step-by Step Solution



- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input x_2
- What should be Bob's input? Let's work out the equation:

$$\begin{aligned} s_{x_2} &= x_2 \cdot s_1 + (1 - x_2) \cdot s_0 \\ &= x_2 \cdot s_1 \oplus (1 \oplus x_2) \cdot s_0 \\ &= s_0 \oplus (s_0 \oplus s_1) \cdot x_2 \end{aligned}$$

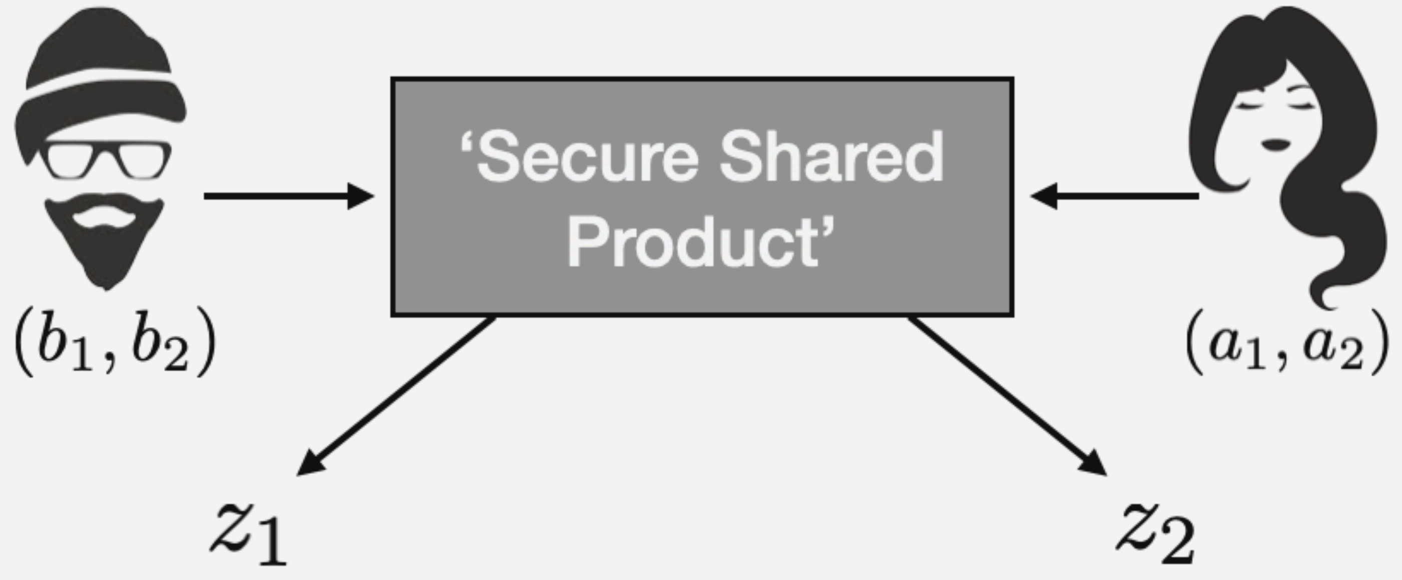
$\Longrightarrow$ $s_0 \oplus s_{x_2} = (s_0 \oplus s_1) \cdot x_2$

Share of Bob — This should be x_1

$\Longrightarrow$ (s_0, s_1) are (2,2)-shares of x_1.

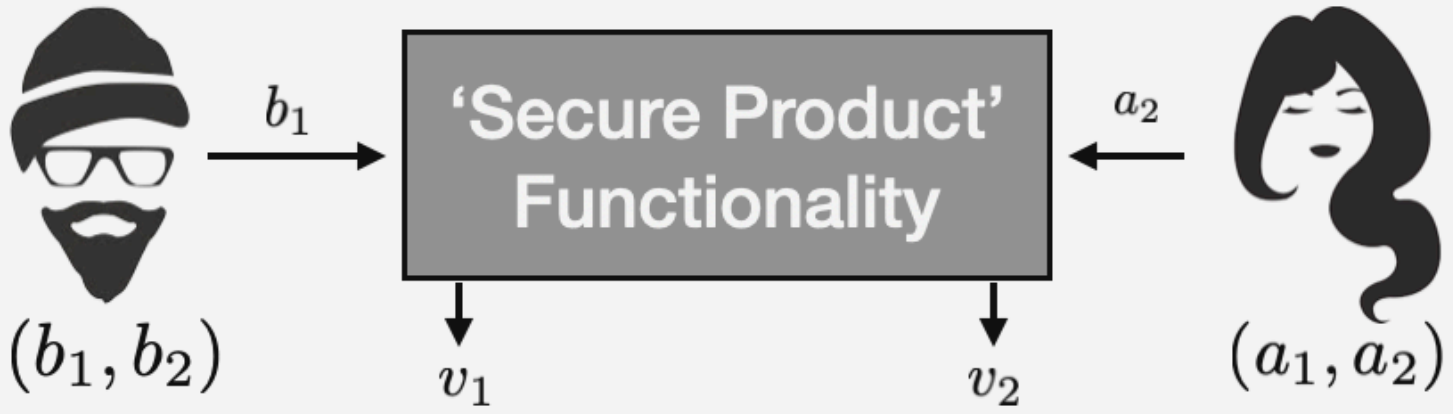
Warm-up II: Variant

This time, Alice and Bob start with *shares* of values (x,y), and want to compute shares of the product x.y



(a_1, b_1) are shares of x
(a_2, b_2) are shares of y
(z_1, z_2) are random shares of $z = x \cdot y$

Solution



$$x \cdot y = (a_1 + b_1) \cdot (a_2 + b_2)$$
$$= a_1 \cdot a_2 + a_1 \cdot b_2 + a_2 \cdot b_1 + b_1 \cdot b_2$$

Value known to Alice — Value known to Bob

Each of these values is the product of a value known to Alice and a value known to Bob