Correlated Pseudorandomness

Achieving faster secure computation through pseudorandom correlation generators

Geoffroy Couteau

Start of the Story: Circa 2016

Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16

Start of the Story: Circa 2016

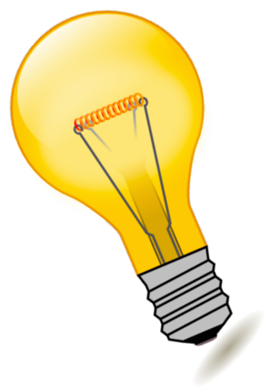
Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16

I was at the time a young 2nd-year PhD student

Start of the Story: Circa 2016

Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16

I was at the time a young 2nd-year PhD student



I got the feeling that this had to be useful to generate correlated randomness…

Start of the Story: Circa 2016

Elette, Niv, and Yuval had just published ‘Breaking the Circuit Size Barrier for Secure Computation under DDH’ at CRYPTO’16

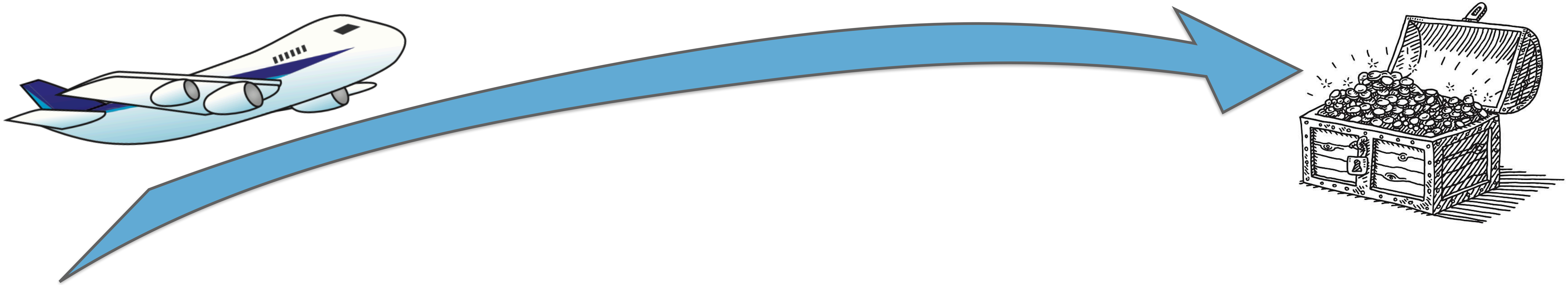
I was at the time a young 2nd-year PhD student

I got the feeling that this had to be useful to generate correlated randomness…

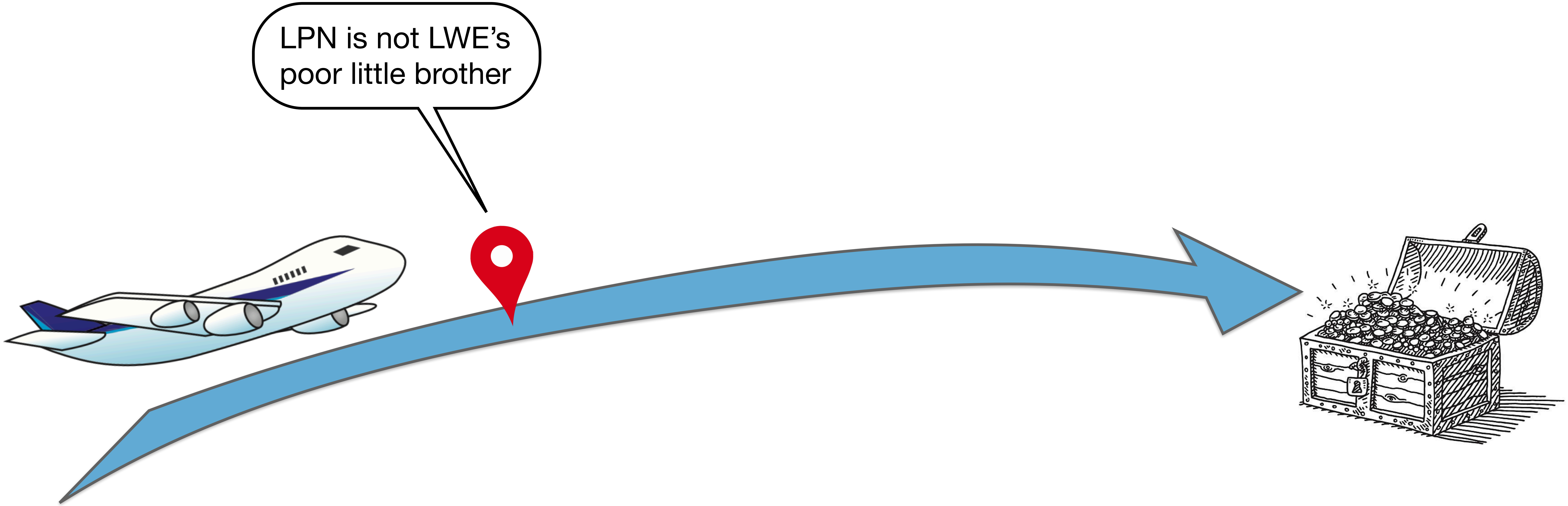
A few months and a CCS’17 paper later, we concluded that the answer was ‘not so much’

7 years later

The Journey through Correlated Pseudorandomess

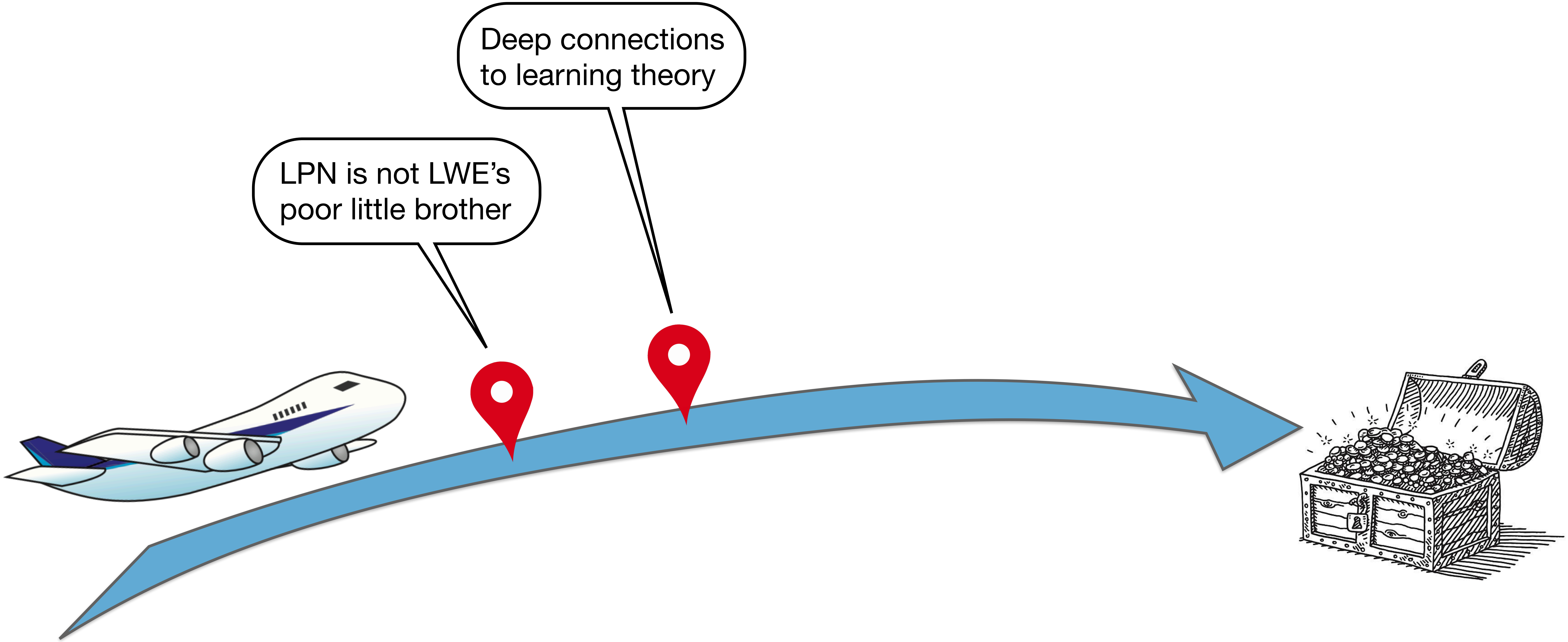


The Journey through Correlated Pseudorandomess



LPN is not LWE's poor little brother

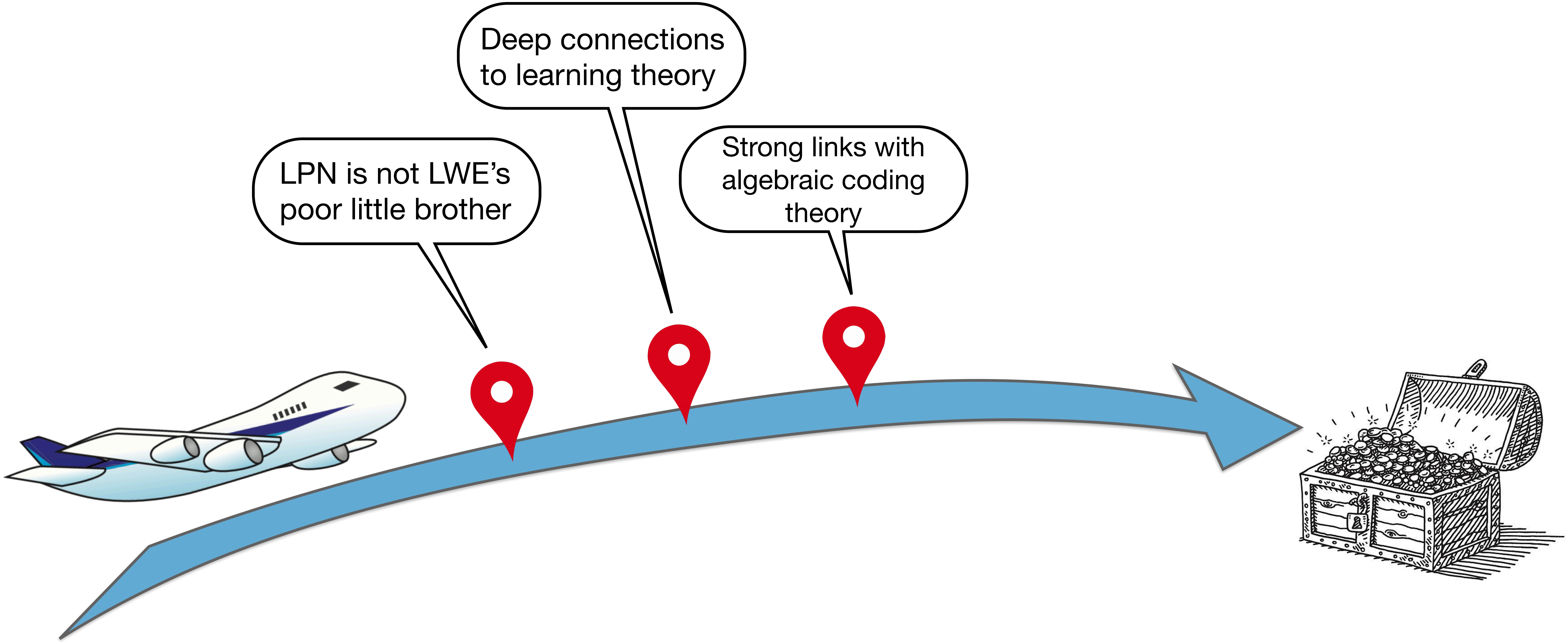
The Journey through Correlated Pseudorandomess



LPN is not LWE’s poor little brother

Deep connections to learning theory

The Journey through Correlated Pseudorandomess

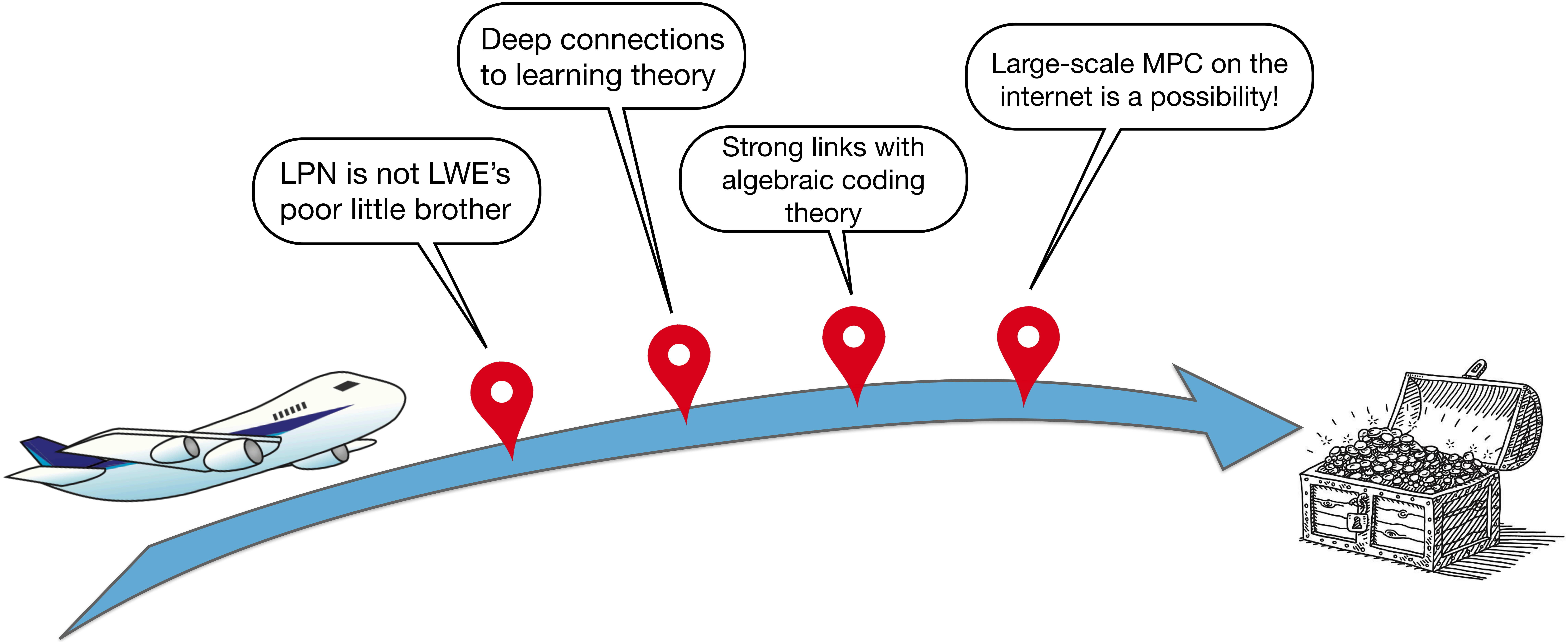


LPN is not LWE’s poor little brother

Deep connections to learning theory

Strong links with algebraic coding theory

The Journey through Correlated Pseudorandomess



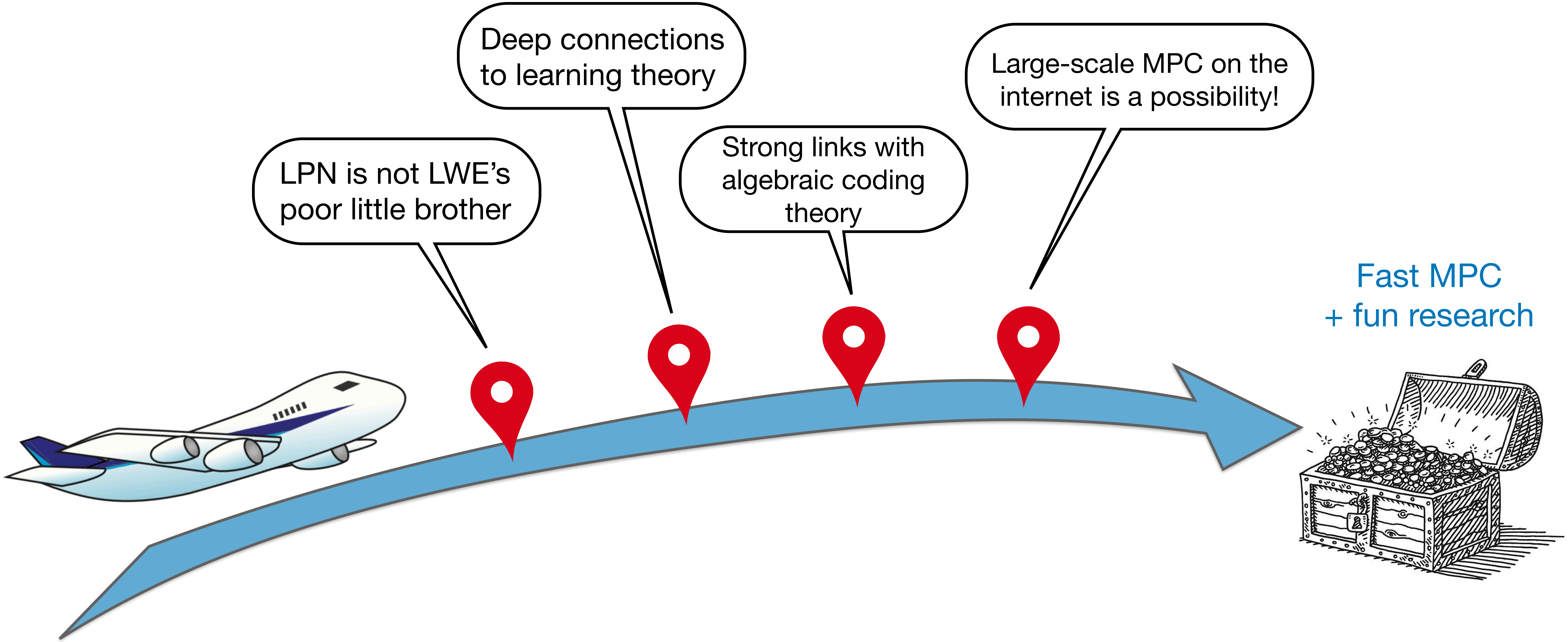
LPN is not LWE’s poor little brother

Deep connections to learning theory

Strong links with algebraic coding theory

Large-scale MPC on the internet is a possibility!

The Journey through Correlated Pseudorandomess



LPN is not LWE’s poor little brother

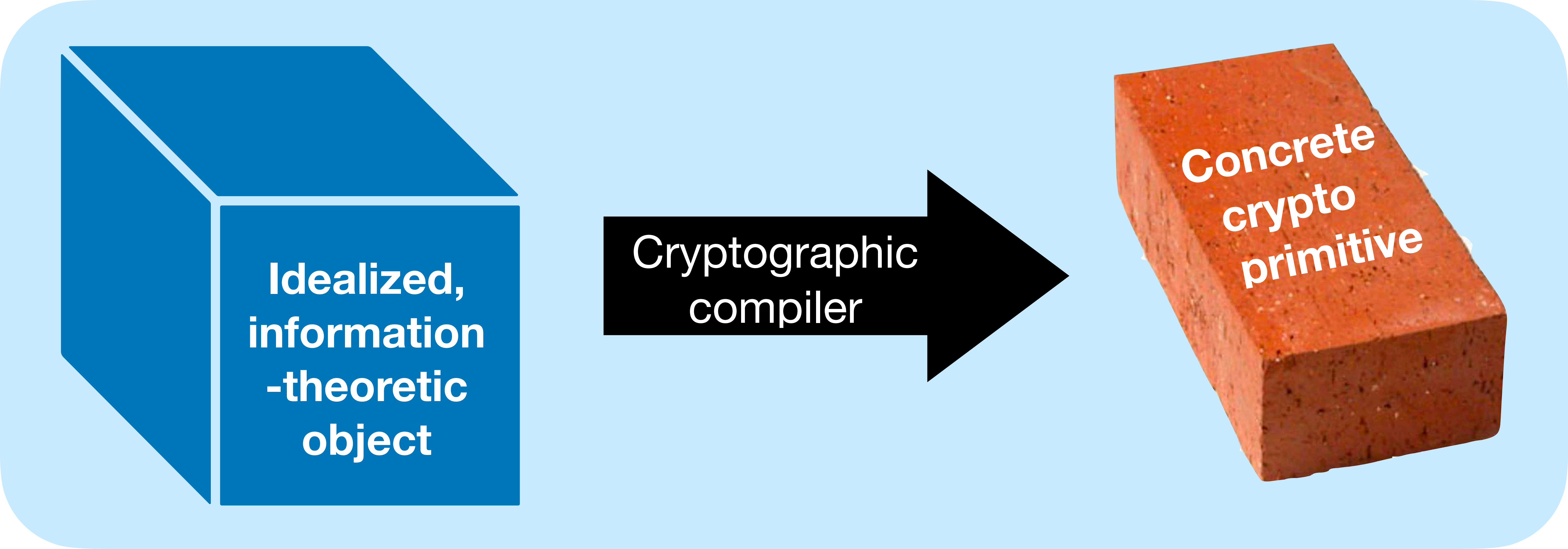
Deep connections to learning theory

Strong links with algebraic coding theory

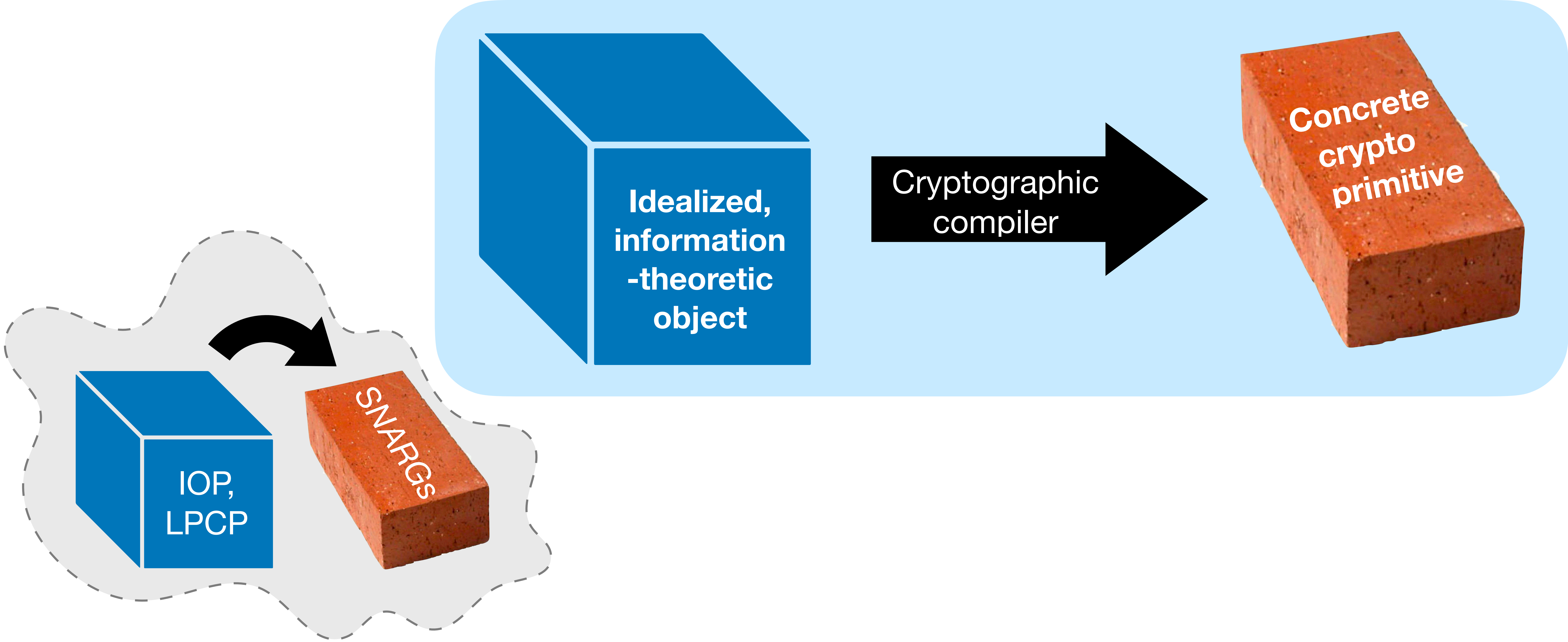
Large-scale MPC on the internet is a possibility!

Fast MPC
+ fun research

A Standard Cryptographic Approach



A Standard Cryptographic Approach



Idealized, information-theoretic object

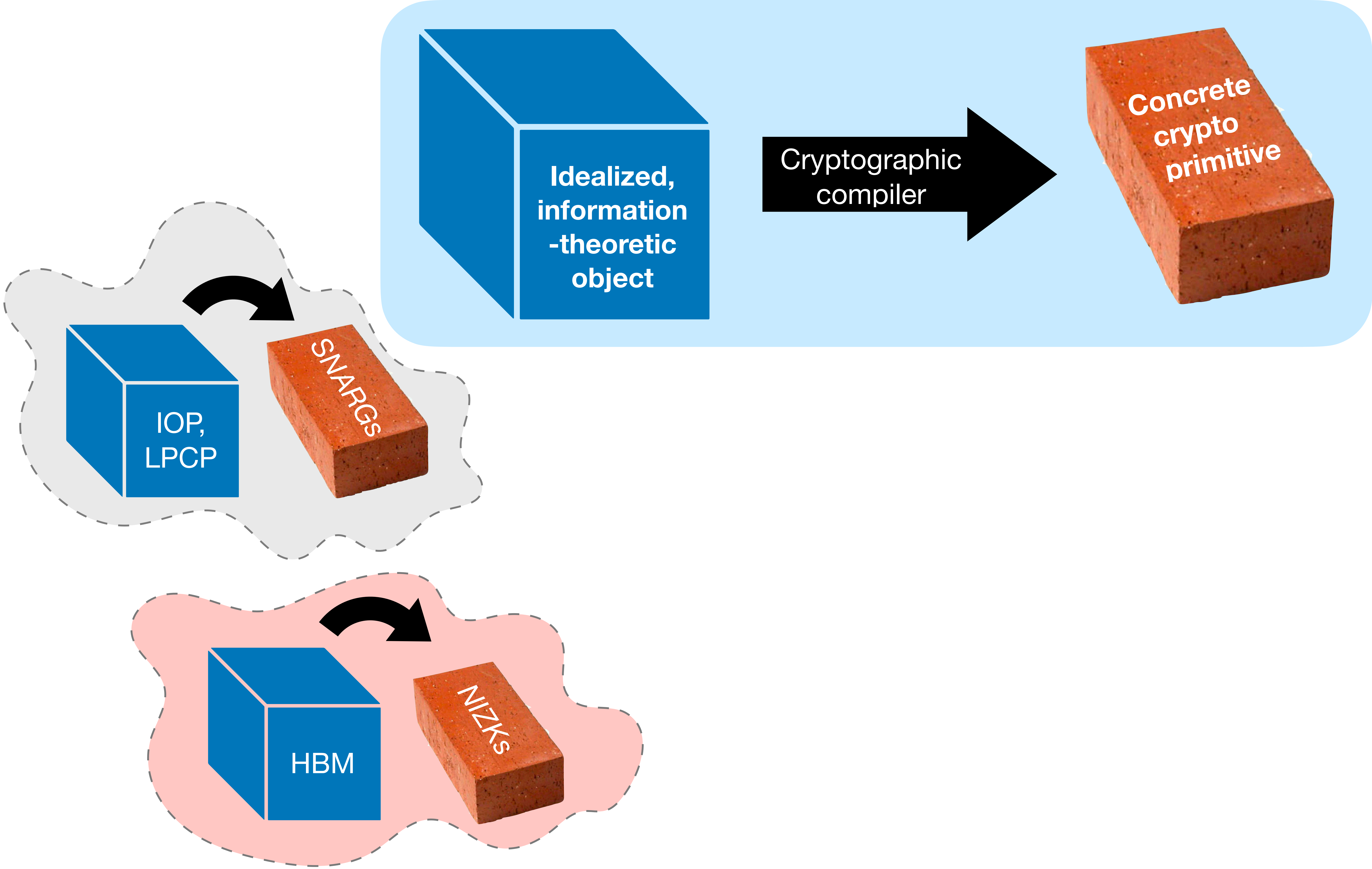
Cryptographic compiler

Concrete crypto primitive

IOP, LPCP

SNARGs

A Standard Cryptographic Approach



Idealized, information-theoretic object

Cryptographic compiler

Concrete crypto primitive

IOP, LPCP

SNARGs

HBM

NIZKs

A Standard Cryptographic Approach

Idealized, information -theoretic object

Cryptographic compiler

Concrete crypto primitive

IOP, LPCP

SNARGs

HBM

NIZKs

OTP

Encryption

A Standard Cryptographic Approach

Idealized,
information
-theoretic
object

Cryptographic
compiler

Concrete
crypto
primitive

IOP,
LPCP

SNARGs

HBM

NIZKs

Correlated
randomness
model

MPC

OTP

Encryption

…

A Standard Cryptographic Approach

Idealized, information-theoretic object

Cryptographic compiler

Concrete crypto primitive

IOP, LPCP

SNARGs

HBM

NIZKs

Correlated randomness model

MPC

OTP

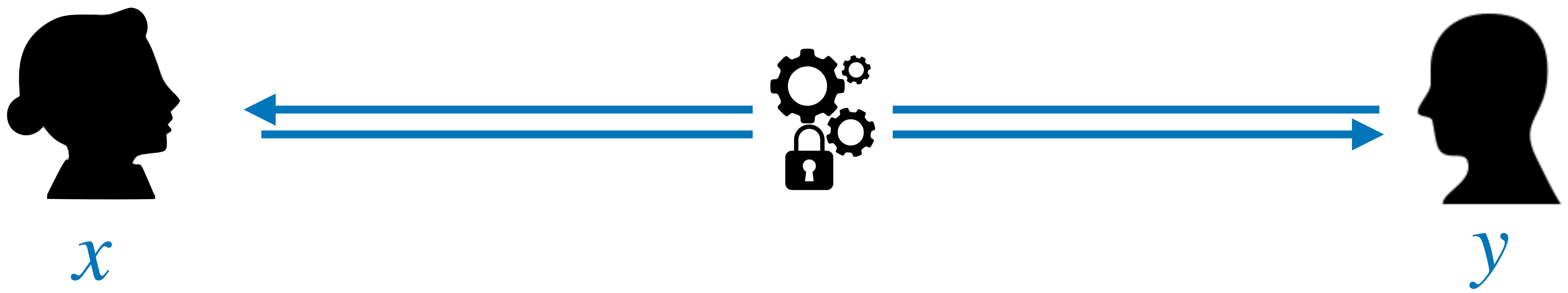
Encryption

• • •

Secure Computation…

- **Goal.** *Computing* a **public** function on secret inputs
- **Model.** n players, each with a private input x_i interacting through authenticated channels

$$\boxed{f:(x,y)\mapsto(z_A,z_B)}$$



- **Output:** Alice learns z_A and Bob learn z_B
- **Security:** Alice and Bob learn nothing else

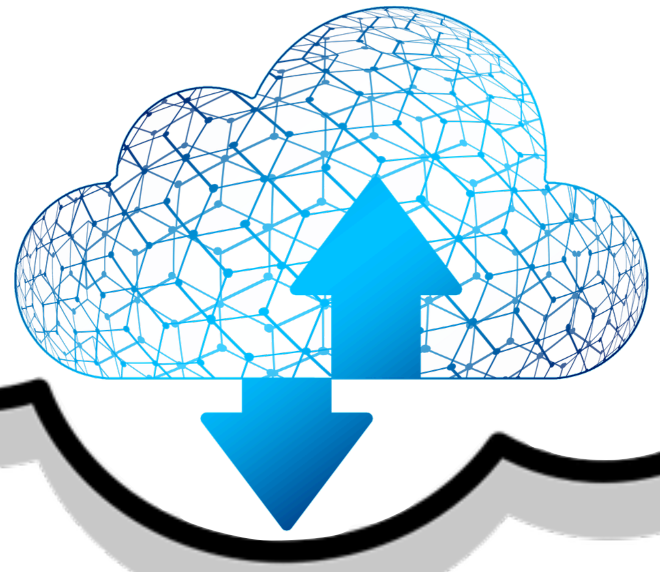
… Is a Practical Concern.

See a targeted advertising

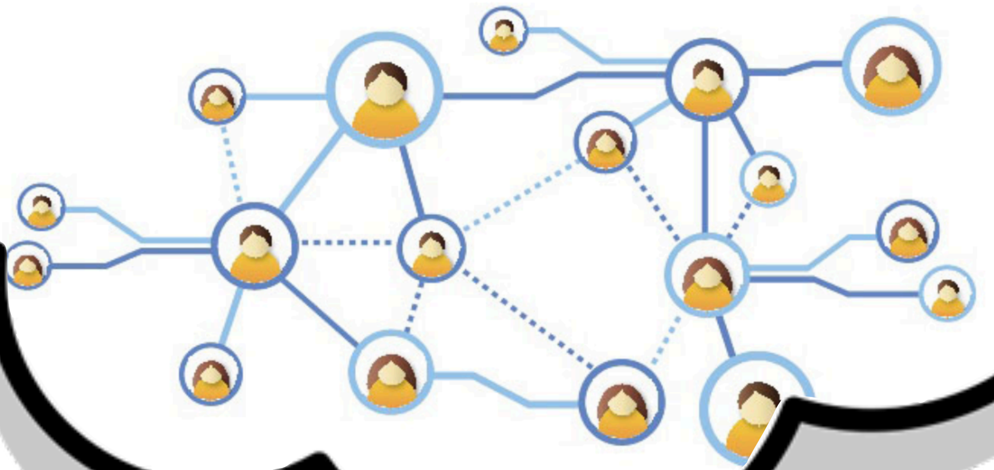
Yes—they're here! Those fashion-wise, comfort-wise Bates Originals—and PHI BATES, the smartest shoes on the campus, style-endorsed by the PHI BATES campus advisory board. Nationally advertised favorites, these shoes bring you an exclusive *extra* comfort—invisible extra width is built right in across the ball of the foot. They're "Slipper-Free Where Your Foot Bends." Drop in today and see the full Bates line.

Use a dating app

Search over our Cloud storage

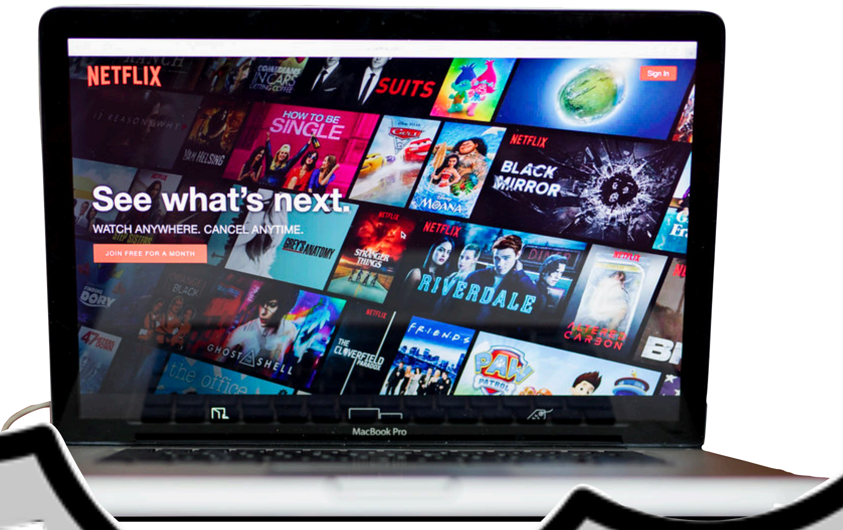


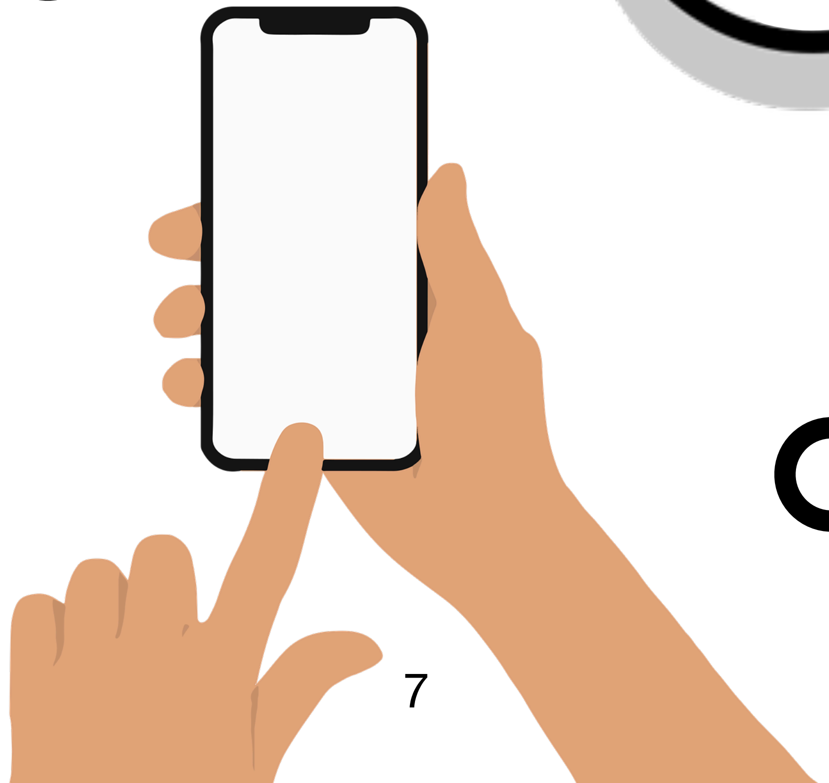
Use a social network



Use a healthcare app

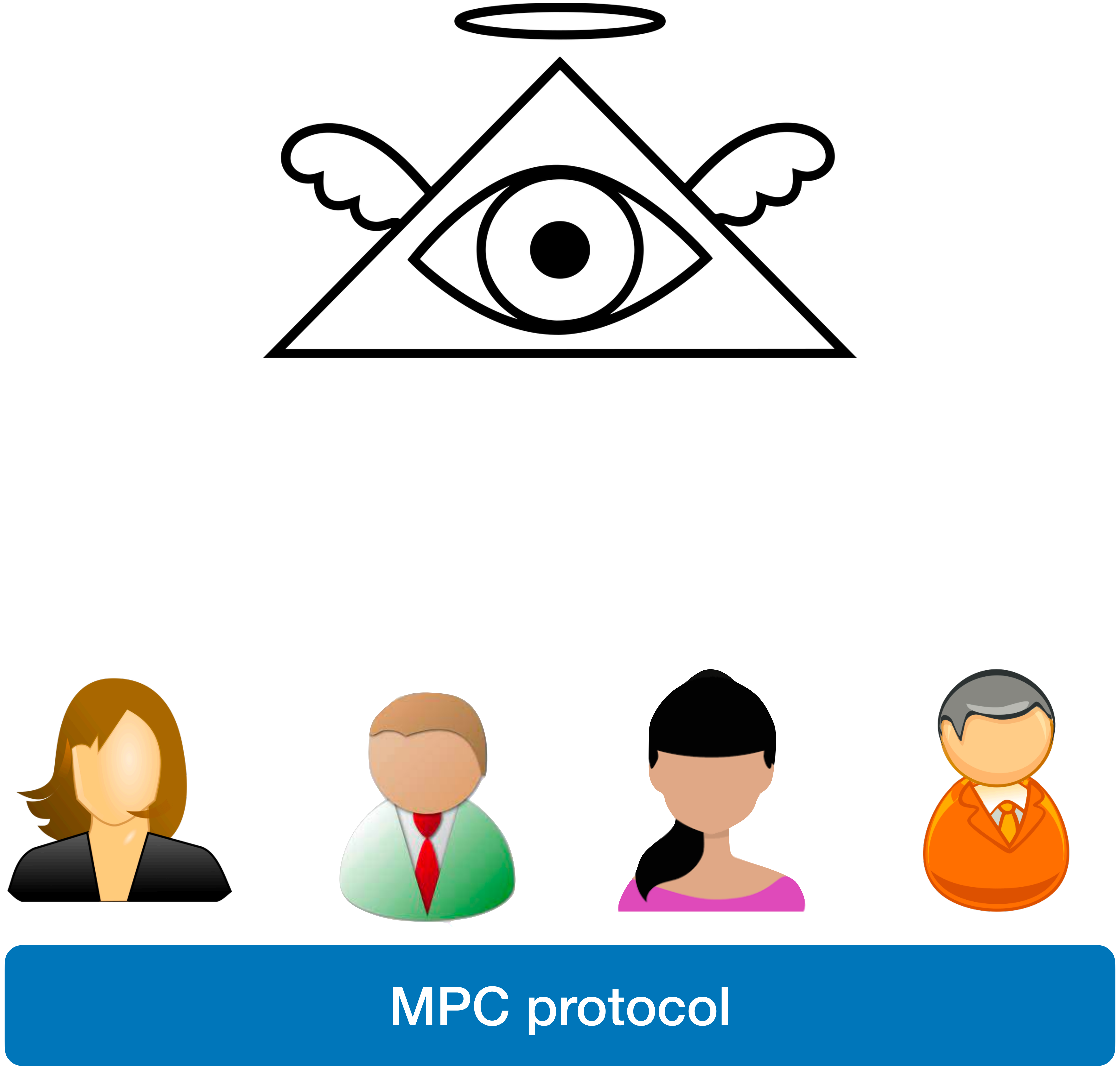
Get a recommendation on a streaming platform





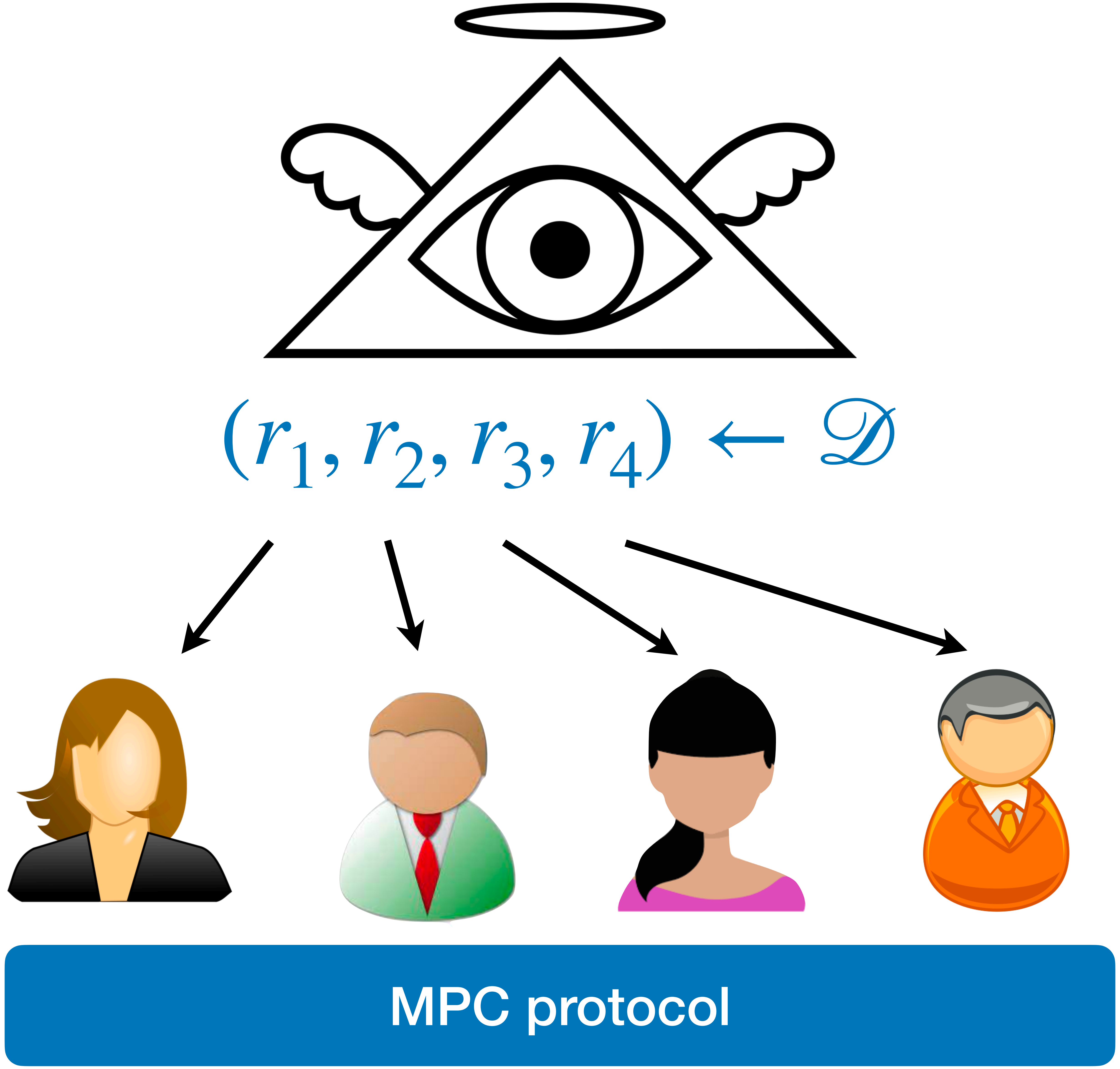
. . .

The Correlated Randomness Model



MPC protocol

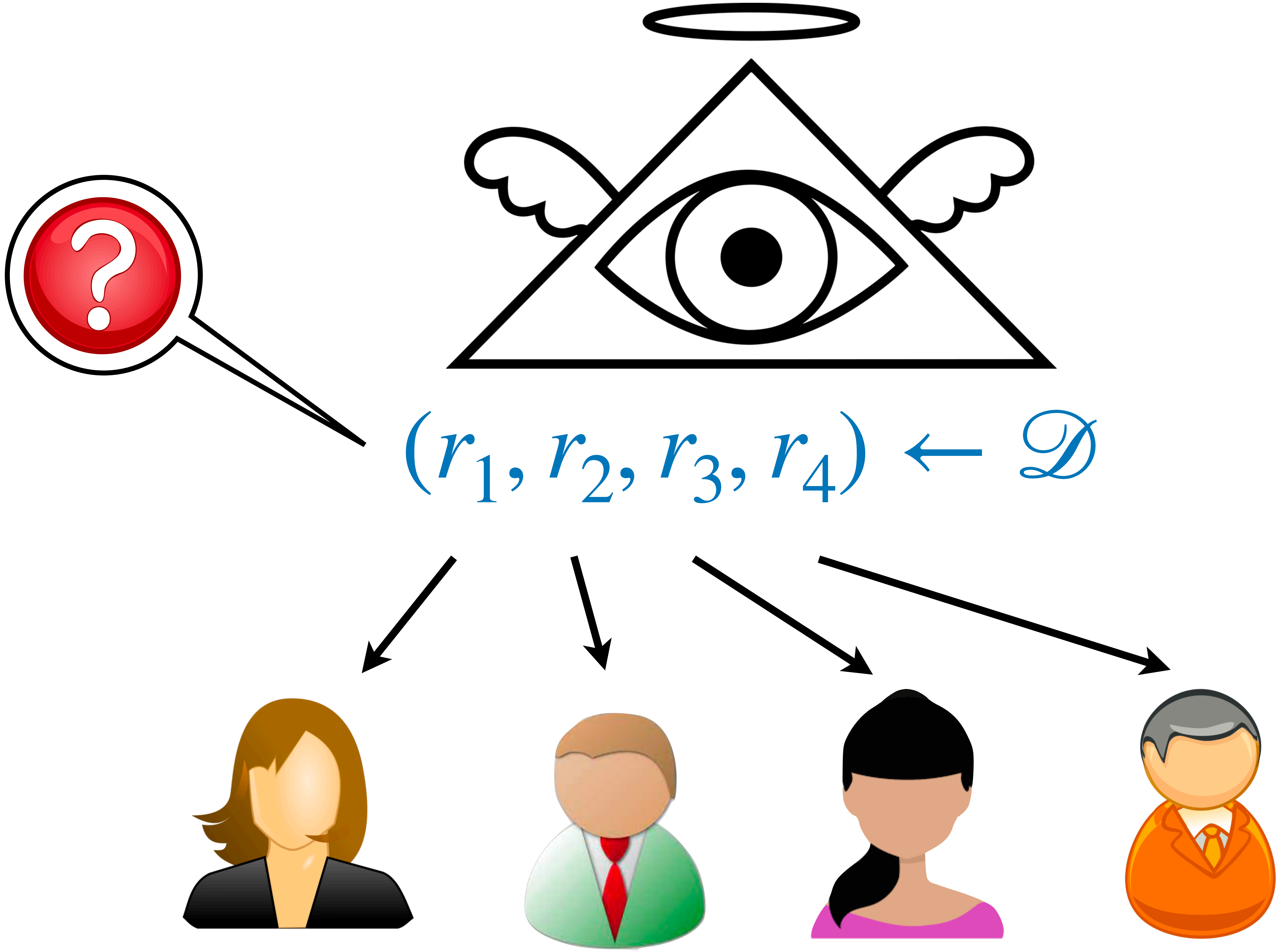
The Correlated Randomness Model



$(r_1, r_2, r_3, r_4) \leftarrow \mathscr{D}$

MPC protocol

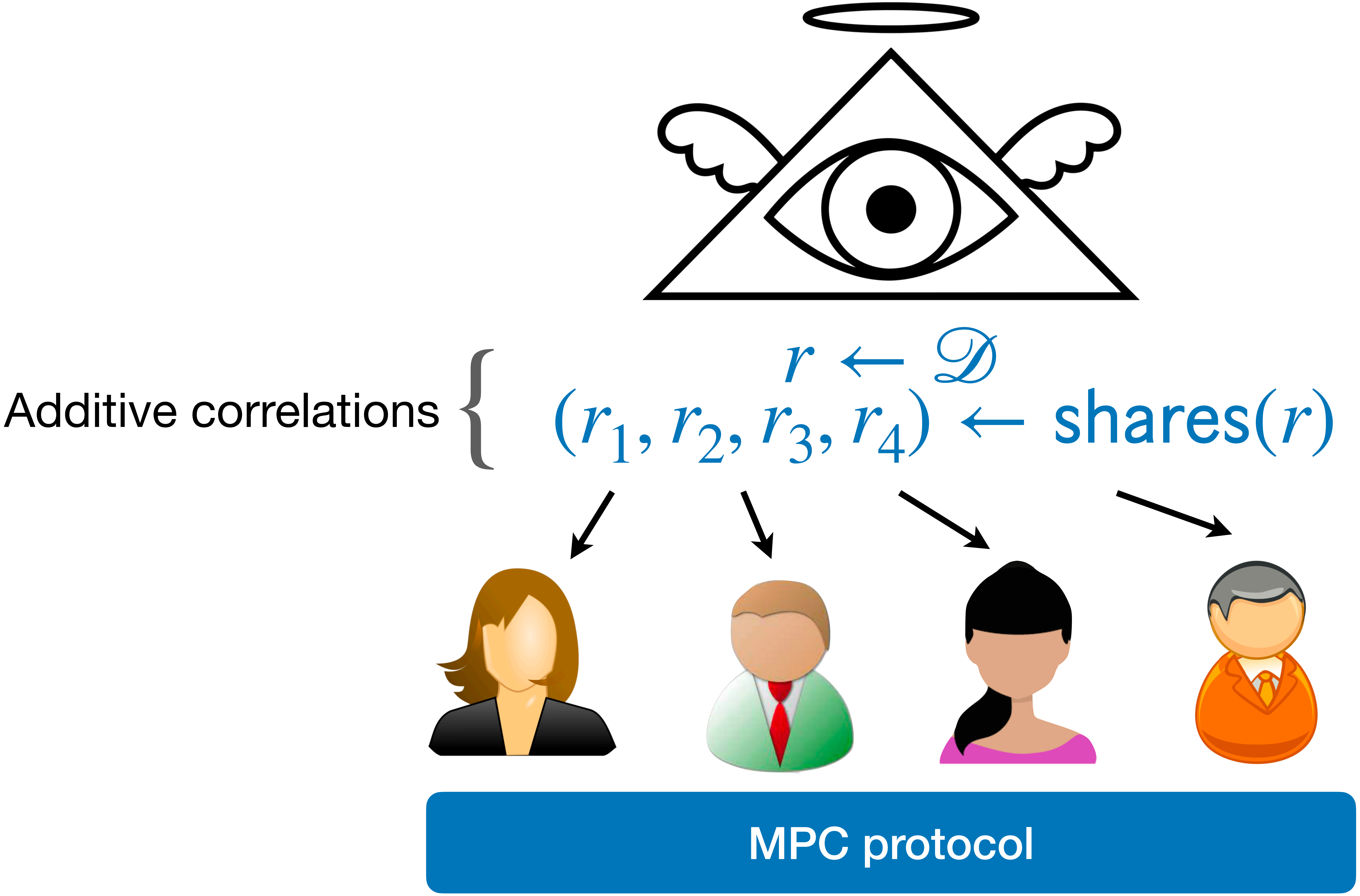
The Correlated Randomness Model



$(r_1, r_2, r_3, r_4) \leftarrow \mathscr{D}$

MPC protocol

The Correlated Randomness Model

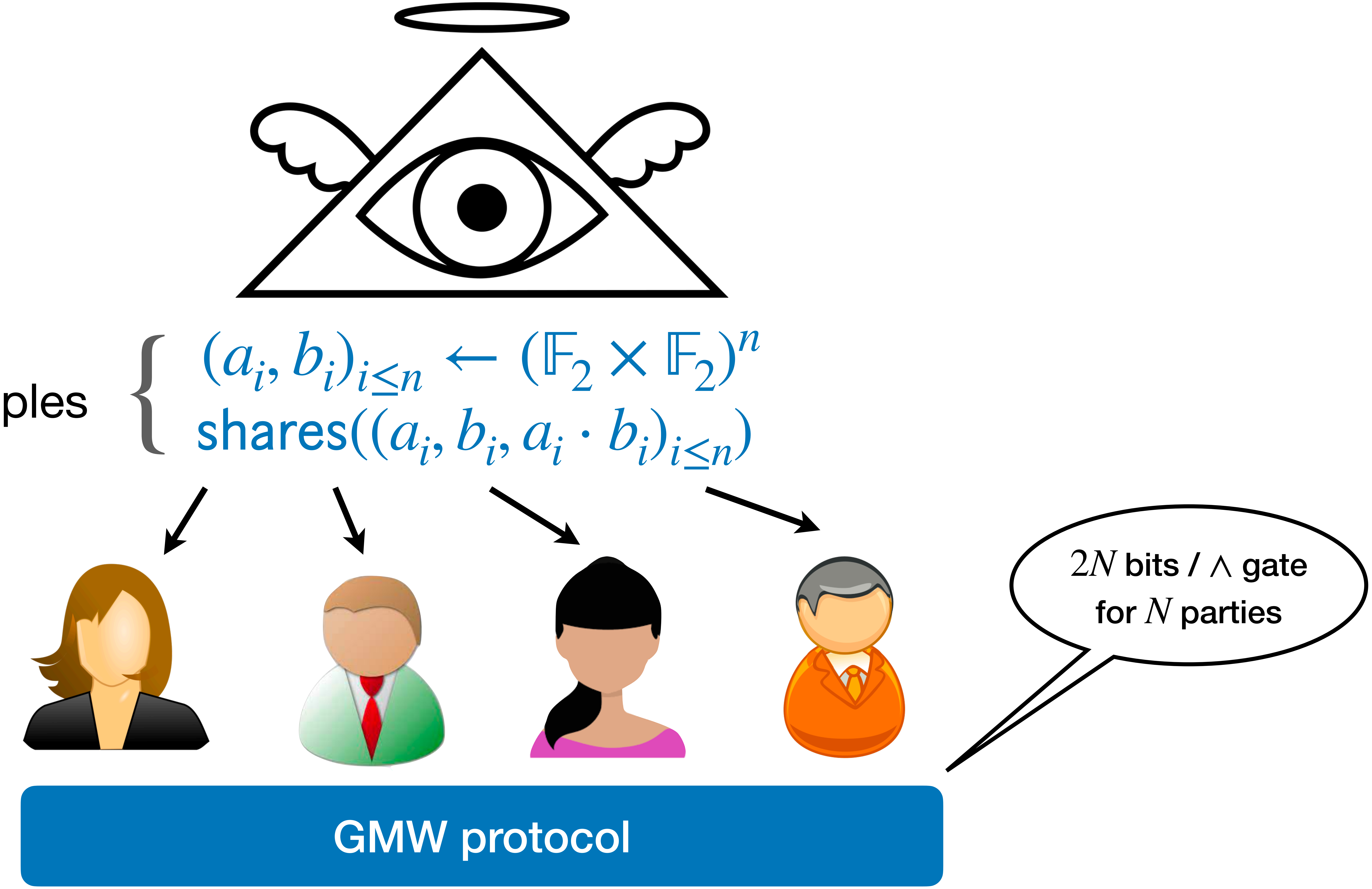


$$r \leftarrow \mathscr{D}$$

Additive correlations $\left\{ \quad (r_1, r_2, r_3, r_4) \leftarrow \mathsf{shares}(r) \right.$

MPC protocol

The Correlated Randomness Model

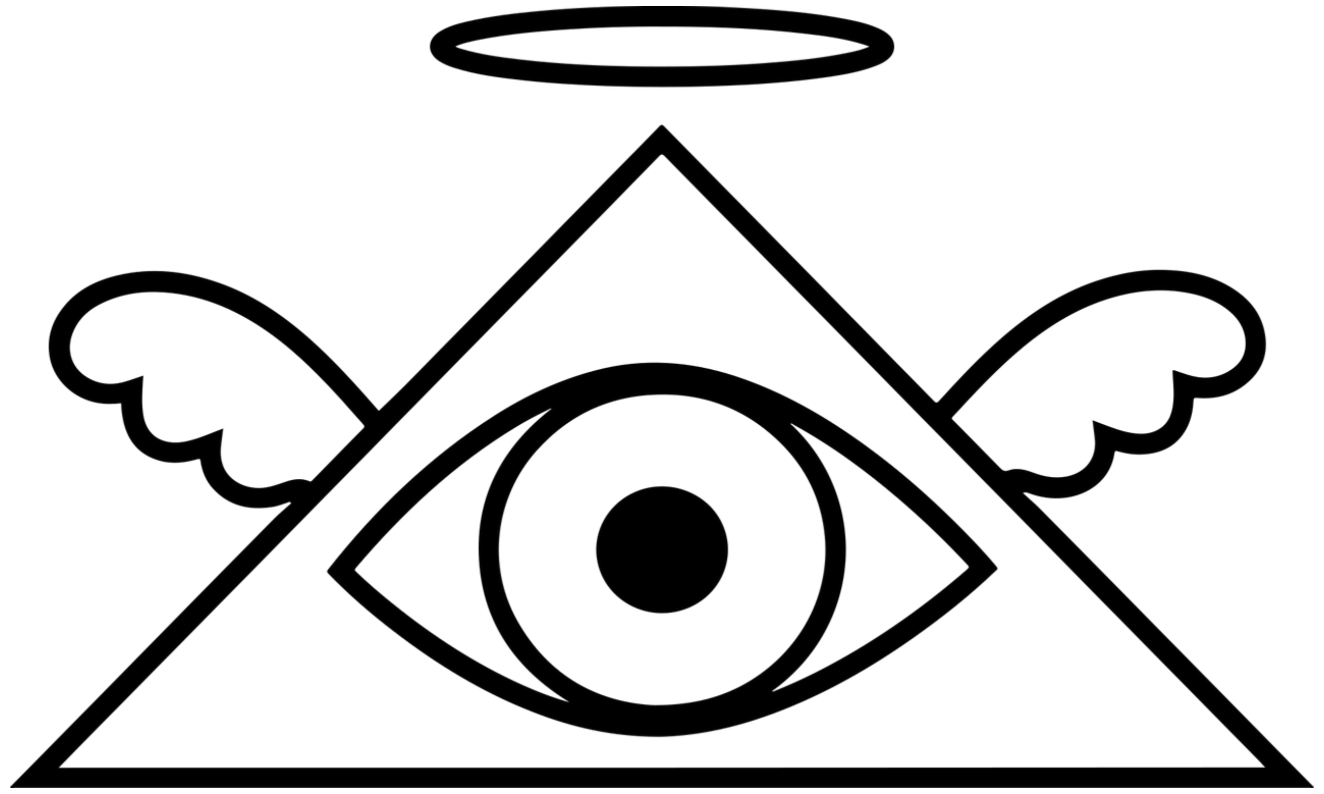


Example: Beaver triples $\left\{ \begin{array}{l} (a_i, b_i)_{i\leq n} \leftarrow (\mathbb{F}_2 \times \mathbb{F}_2)^n \\ \text{shares}((a_i, b_i, a_i \cdot b_i)_{i\leq n}) \end{array} \right.$

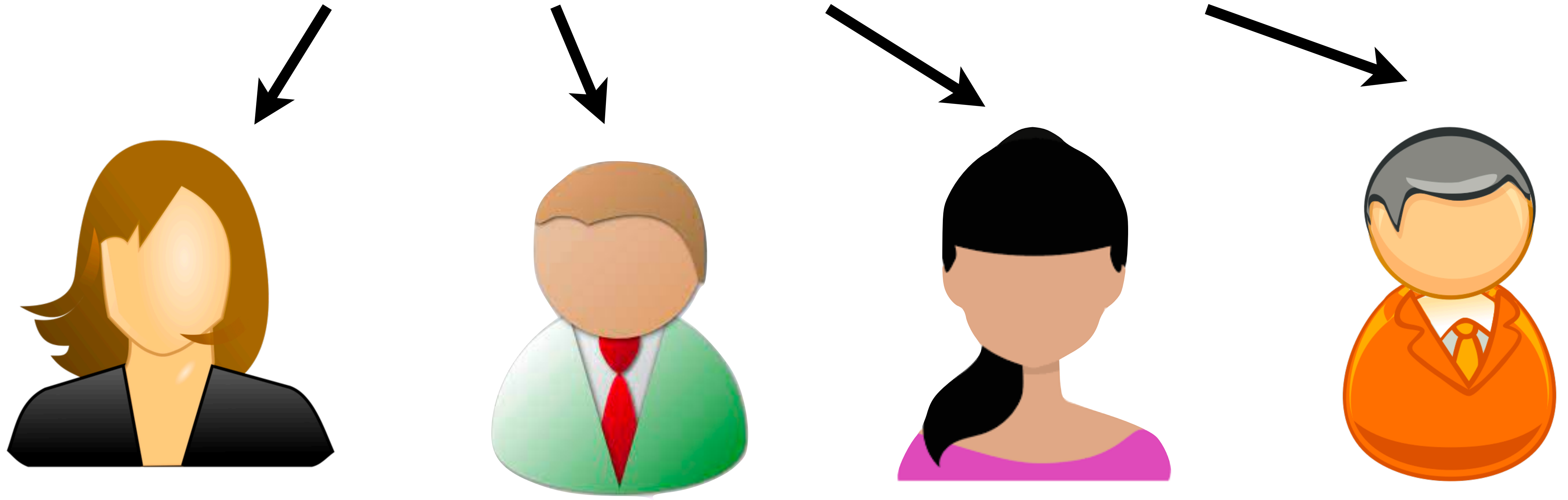
GMW protocol

$2N$ bits / $\wedge$ gate
for N parties

The Correlated Randomness Model



Example: Beaver triples $\begin{cases} (a_i, b_i)_{i \leq n} \leftarrow (\mathbb{F}_2 \times \mathbb{F}_2)^n \\ \text{shares}((a_i, b_i, a_i \cdot b_i)_{i \leq n}) \end{cases}$



GMW protocol

$2N$ bits / $\wedge$ gate
for N parties

Very practical*

A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation C, the dealer distributes shares of $C(r)$

🎯 Distributing $\langle C(r) \rangle$ *succinctly*

A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation C, the dealer distributes shares of $C(r)$

🎯 Distributing $\langle C(r) \rangle$ *succinctly*

Pseudorandom correlation generator

$\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{seed}_A, \mathsf{seed}_B)$ such that

(1) $(\mathsf{Expand}(A, \mathsf{seed}_A), \mathsf{Expand}(B, \mathsf{seed}_B))$ looks like n samples from the target correlation, and

(2) $\mathsf{Expand}(A, \mathsf{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\mathsf{Expand}(B, \mathsf{seed}_B)$’ to Bob (similar property w.r.t. Alice).

A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

Pseudorandom correlation generator: $\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{seed}_A, \mathsf{seed}_B)$ **such that (1)** $(\mathsf{Expand}(A, \mathsf{seed}_A), \mathsf{Expand}(B, \mathsf{seed}_B))$ **looks like** n **samples from the target correlation, and (2)** $\mathsf{Expand}(A, \mathsf{seed}_A)$ **looks 'random conditioned on satisfying the correlation with** $\mathsf{Expand}(B, \mathsf{seed}_B)$**' to Bob (similar property w.r.t. Alice).**

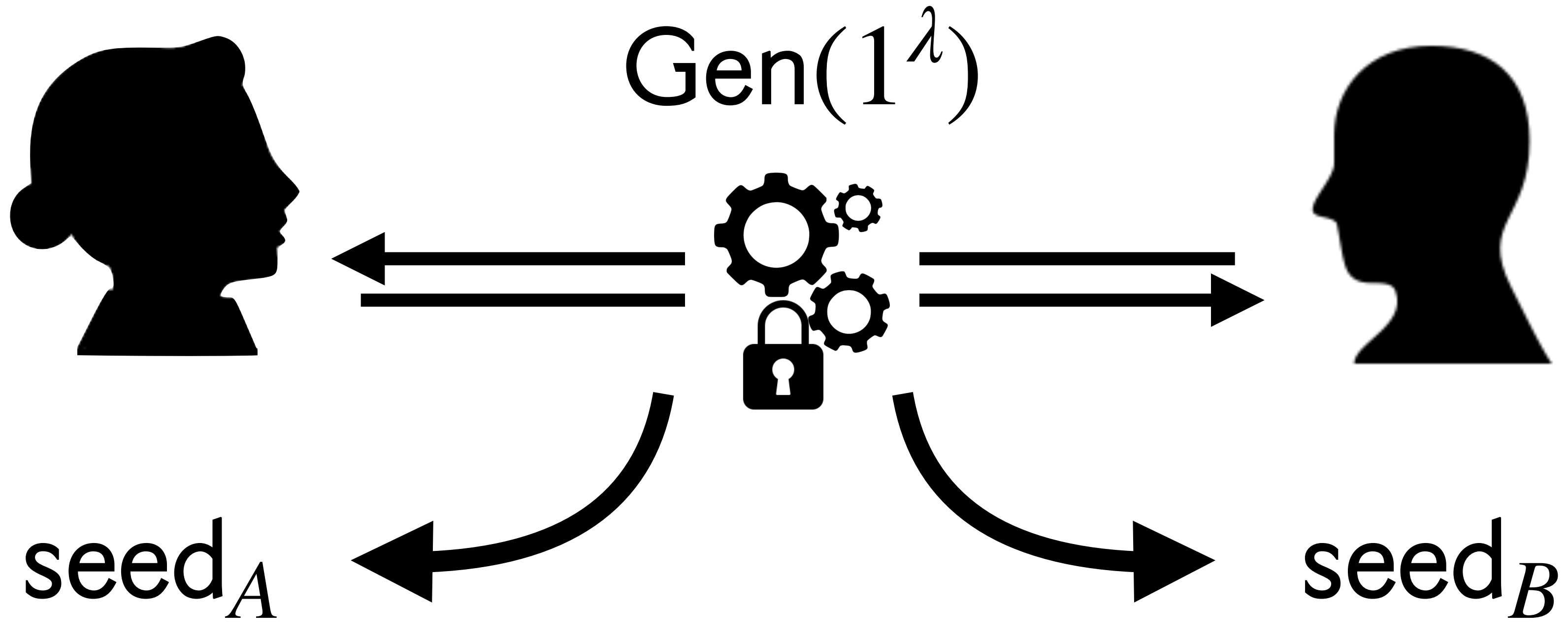
Preprocessing phase

Online phase

A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

Pseudorandom correlation generator: Gen$(1^\lambda) \rightarrow ($seed$_A$, seed$_B)$ **such that (1)** $($Expand$(A,$ seed$_A)$, Expand$(B,$ seed$_B))$ **looks like n samples from the target correlation, and (2)** Expand$(A,$ seed$_A)$ **looks 'random conditioned on satisfying the correlation with** Expand$(B,$ seed$_B)$**' to Bob (similar property w.r.t. Alice).**

One-time short interaction



Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

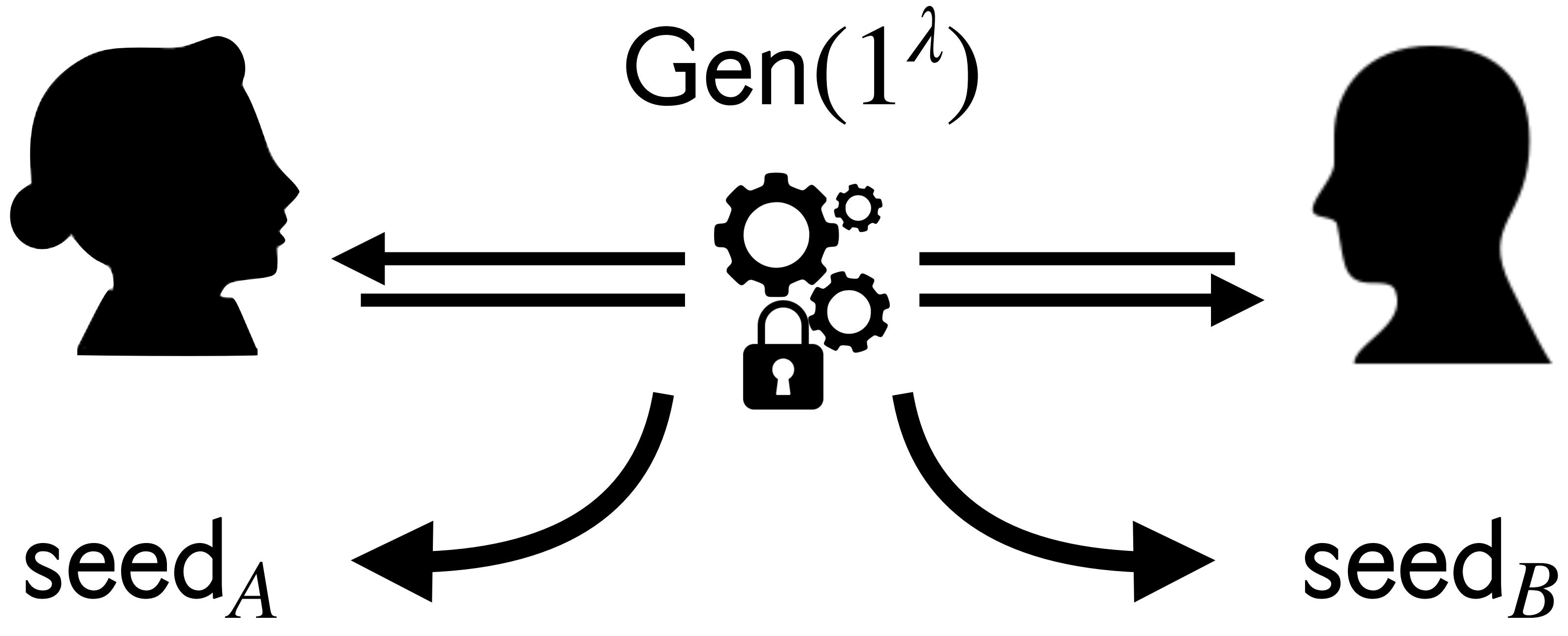
Preprocessing phase

Online phase

A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

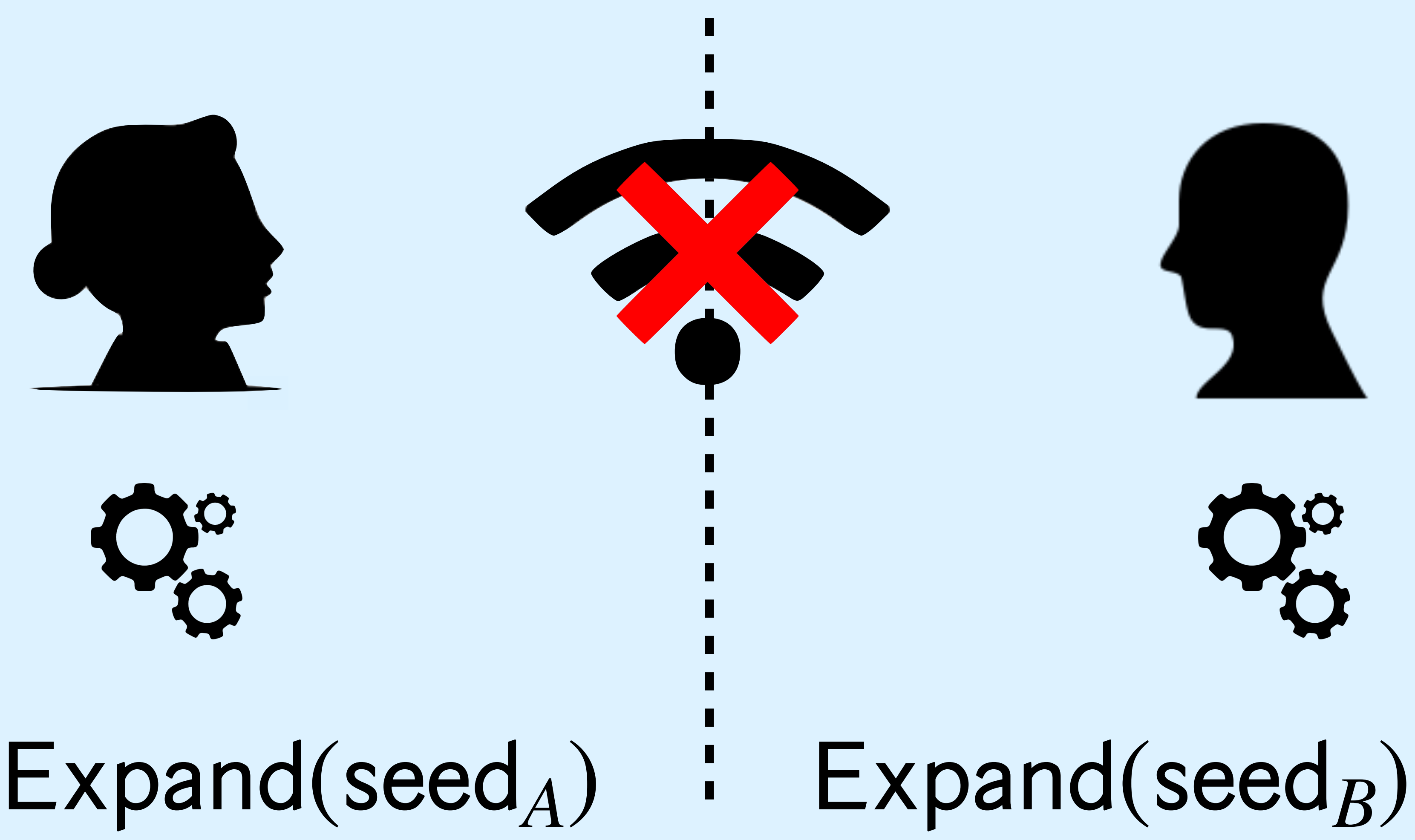
Pseudorandom correlation generator: $\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{seed}_A, \mathsf{seed}_B)$ such that (1) $(\mathsf{Expand}(A, \mathsf{seed}_A), \mathsf{Expand}(B, \mathsf{seed}_B))$ looks like n samples from the target correlation, and (2) $\mathsf{Expand}(A, \mathsf{seed}_A)$ looks 'random conditioned on satisfying the correlation with $\mathsf{Expand}(B, \mathsf{seed}_B)$' to Bob (similar property w.r.t. Alice).

One-time short interaction



Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

'Silent' computation



The bulk of the preprocessing phase is offline: Alice and Bob stretch their seeds into large pseudorandom correlated strings.

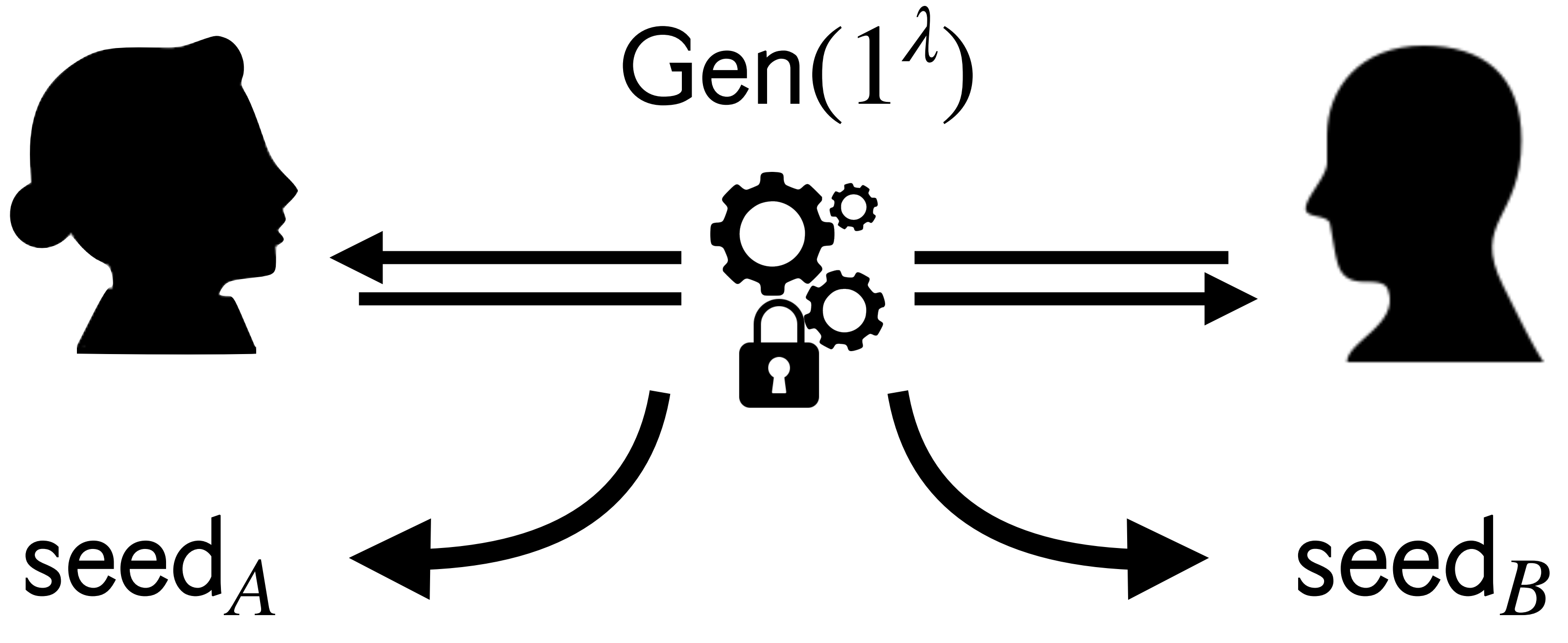
Preprocessing phase

Online phase

A Template to Instantiate *Efficiently* the Correlated Randomness Model: MPC with silent preprocessing

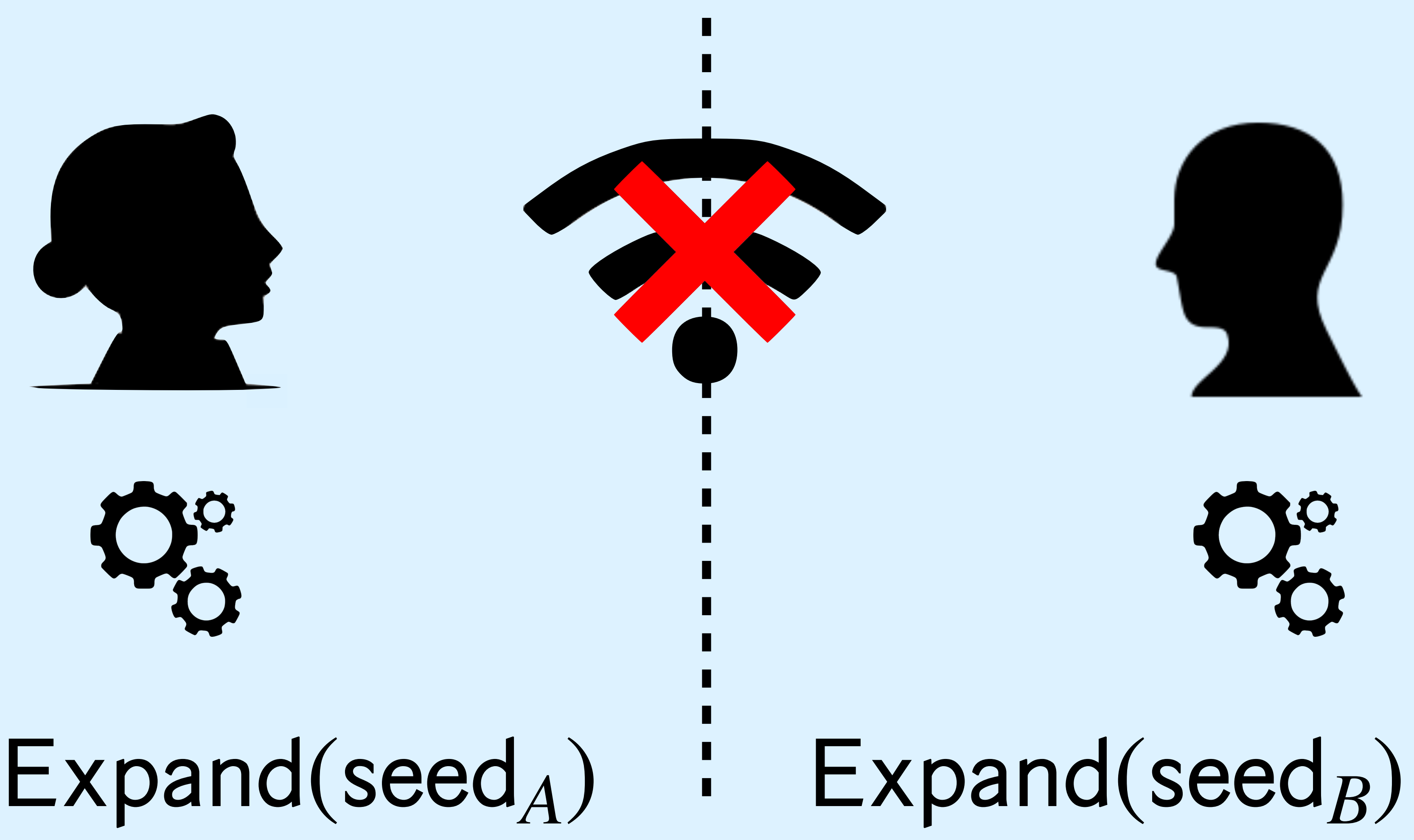
Pseudorandom correlation generator: $\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{seed}_A, \mathsf{seed}_B)$ **such that (1)** $(\mathsf{Expand}(A, \mathsf{seed}_A), \mathsf{Expand}(B, \mathsf{seed}_B))$ **looks like** n **samples from the target correlation, and (2)** $\mathsf{Expand}(A, \mathsf{seed}_A)$ **looks 'random conditioned on satisfying the correlation with** $\mathsf{Expand}(B, \mathsf{seed}_B)$**' to Bob (similar property w.r.t. Alice).**

One-time short interaction



Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.

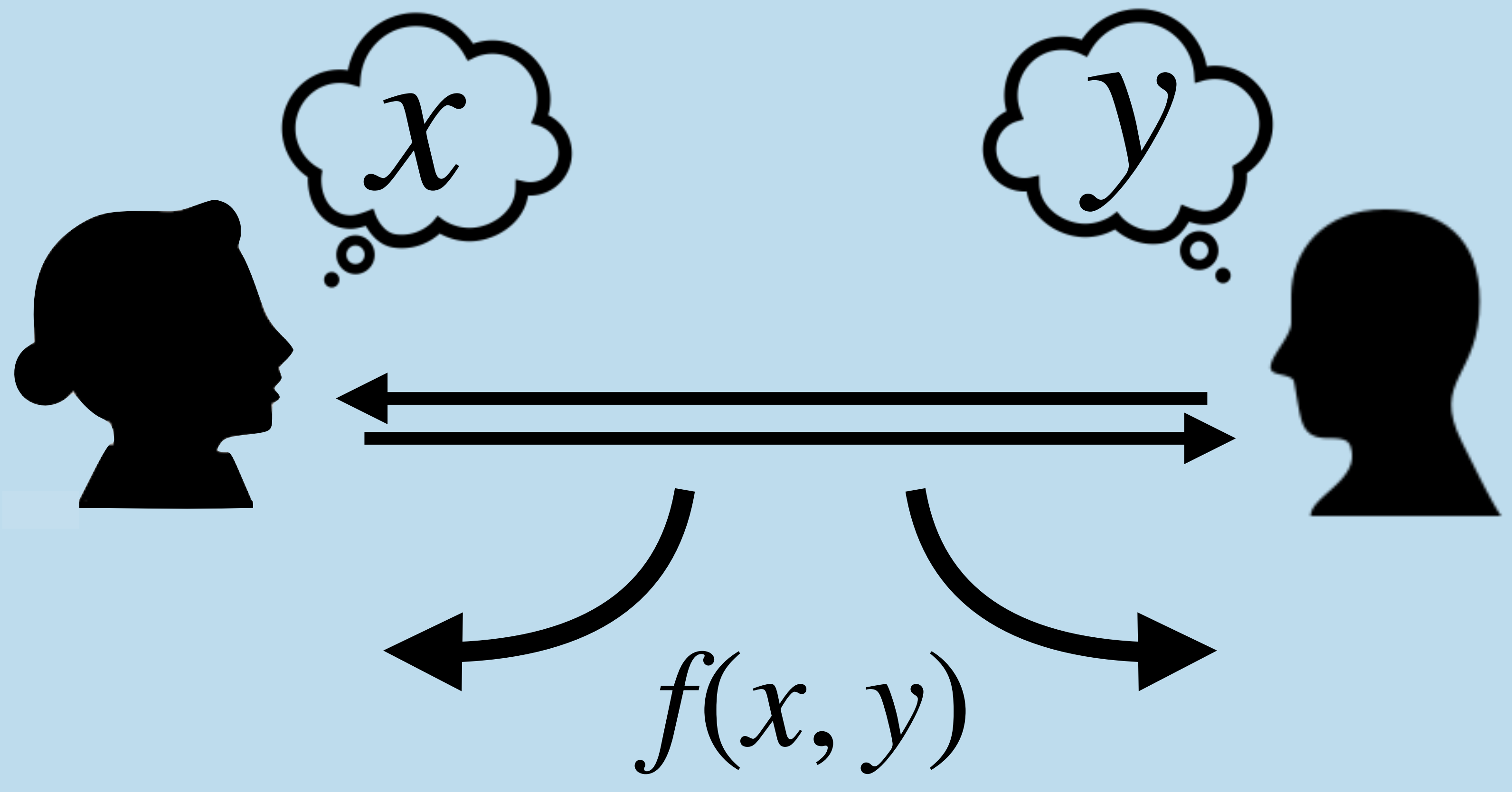
'Silent' computation



The bulk of the preprocessing phase is offline: Alice and Bob stretch their seeds into large pseudorandom correlated strings.

Preprocessing phase

Non-cryptographic



Alice and Bob consume the preprocessing material in a fast, non-cryptographic online phase.

Online phase

Q: What Correlations C do we Consider?

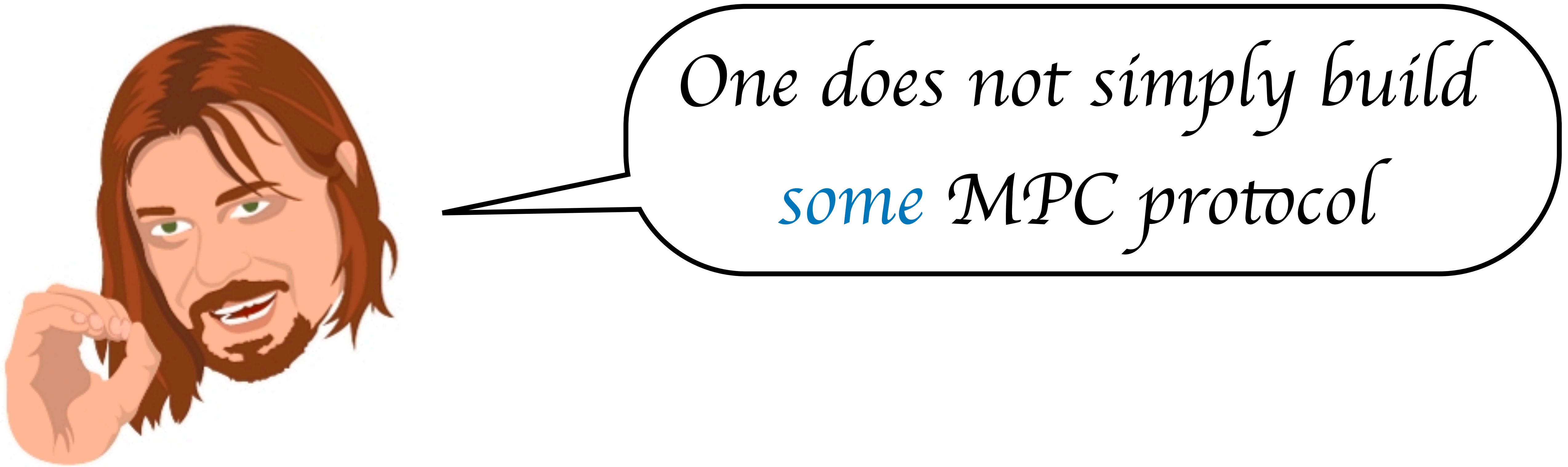
Q: What Correlations C do we Consider?

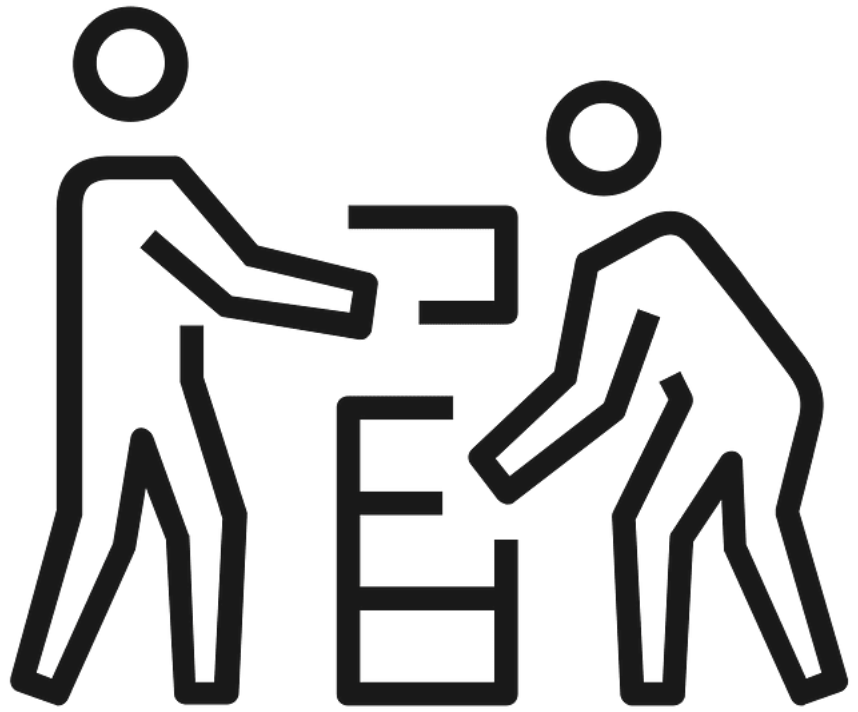
R: It depends! What MPC Protocol do you Want?

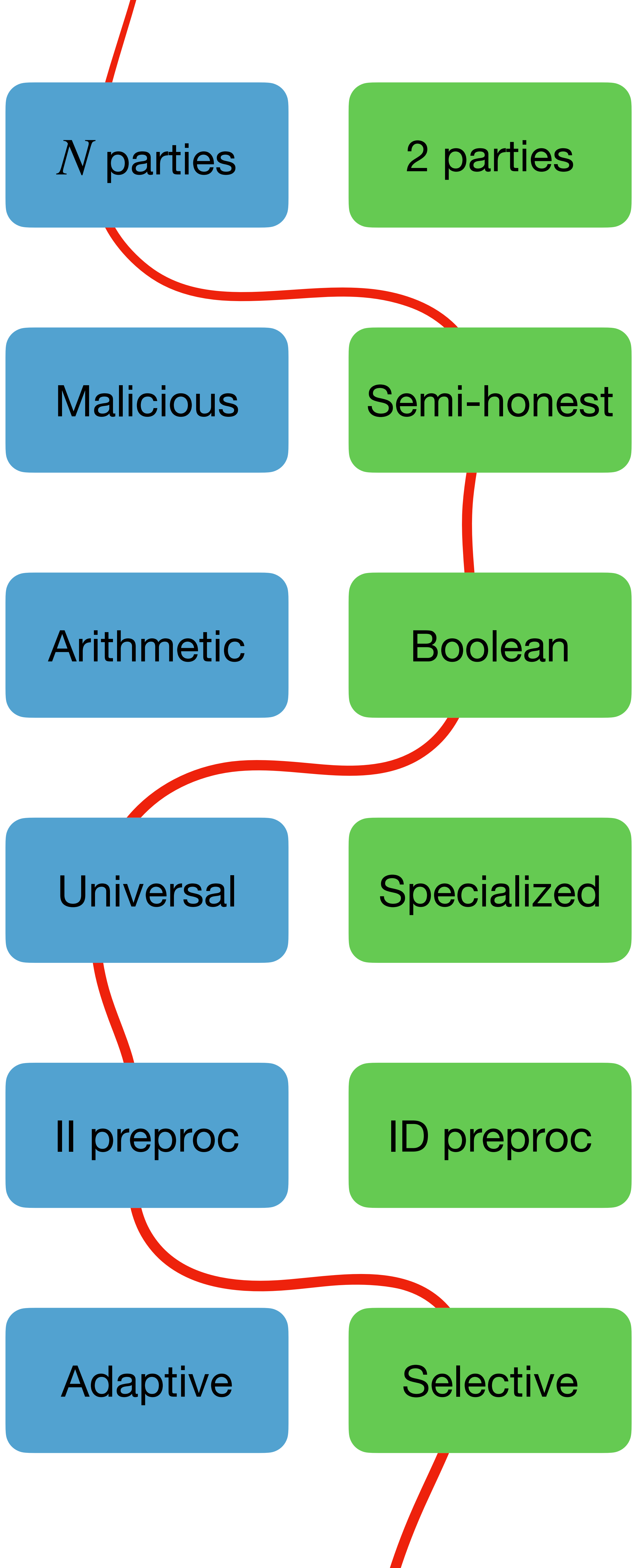
One does not simply build some MPC protocol

Q: What Correlations C do we Consider?

R: It depends! What MPC Protocol do you Want?



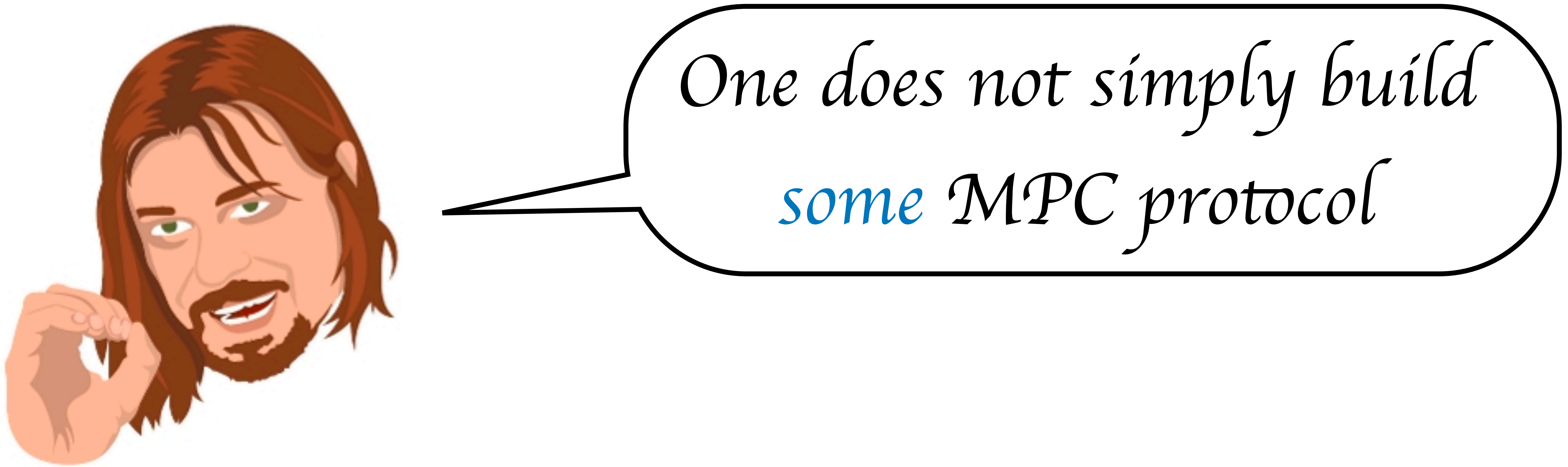




N parties	2 parties
Malicious	Semi-honest
Arithmetic	Boolean
Universal	Specialized
II preproc	ID preproc
Adaptive	Selective

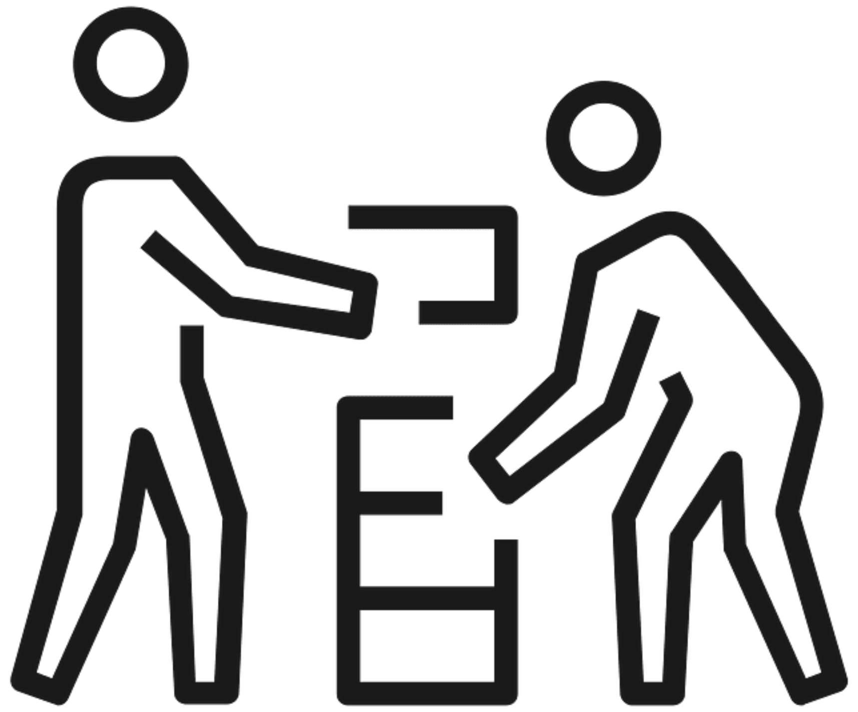
Q: What Correlations C do we Consider?

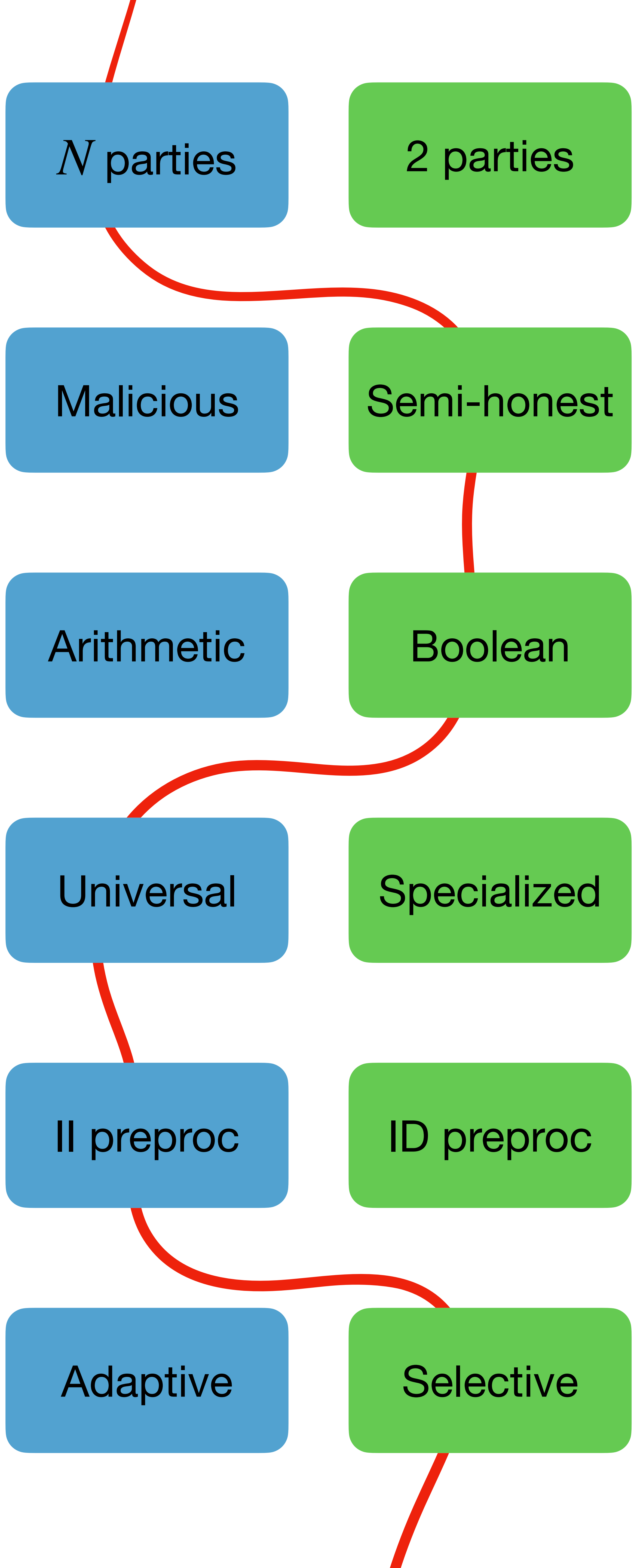
R: It depends! What MPC Protocol do you Want?



Depending on the application, you'll want:

VOLE, OT, OLE, bilinear correlations, Beaver triples, authenticated Beaver triples, daBits, circuit-dependent correlations, polynomial correlations, matrix triples, OTTT…




N parties
2 parties
Malicious
Semi-honest
Arithmetic
Boolean
Universal
Specialized
II preproc
ID preproc
Adaptive
Selective

A Template to Instantiate *Efficiently* the Correlated Randomness Model

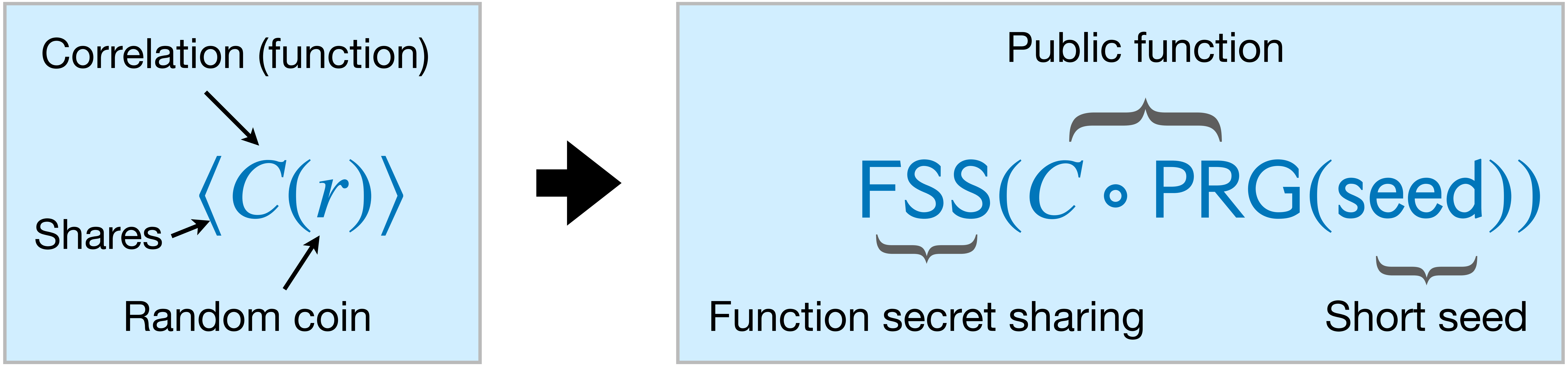
Given a correlation C, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ *succinctly*

A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation C, the dealer distributes shares of $C(r)$

Distributing $\langle C(r) \rangle$ *succinctly*



Correlation (function)

Shares

$$\langle C(r) \rangle$$

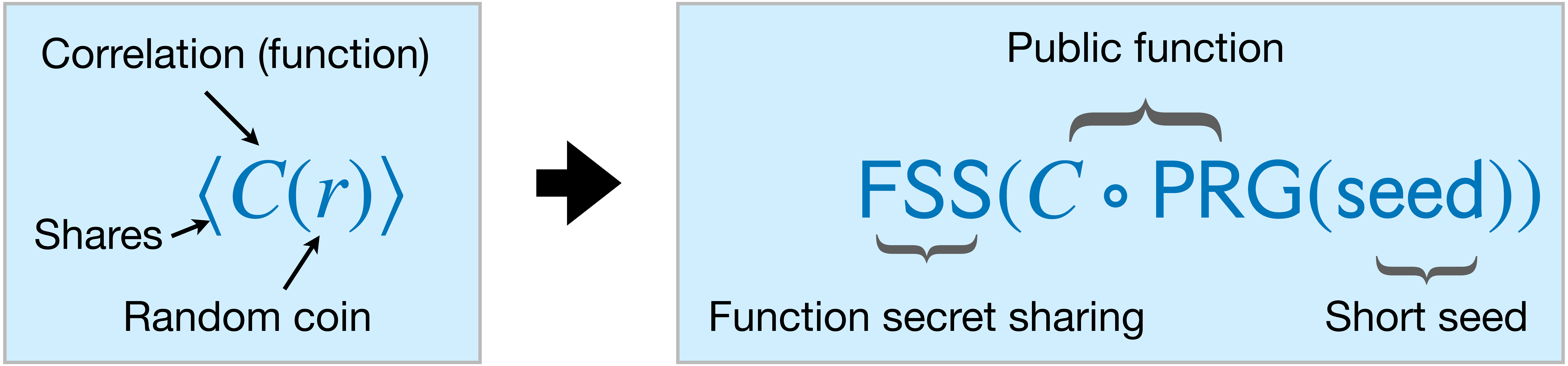
Random coin

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRG}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

A Template to Instantiate *Efficiently* the Correlated Randomness Model

Given a correlation C, the dealer distributes shares of $C(r)$

Distributing $\langle C(r)\rangle$ *succinctly*



Correlation (function)

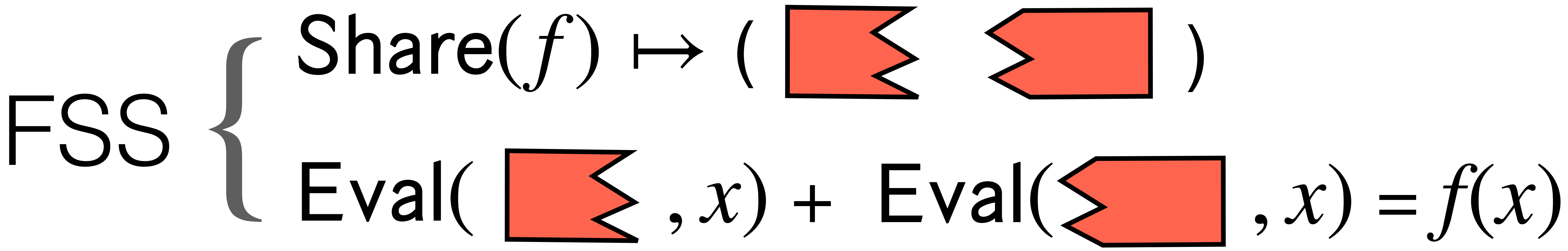
Shares

$\langle C(r)\rangle$

Random coin

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C\circ\mathsf{PRG}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

Function Secret Sharing

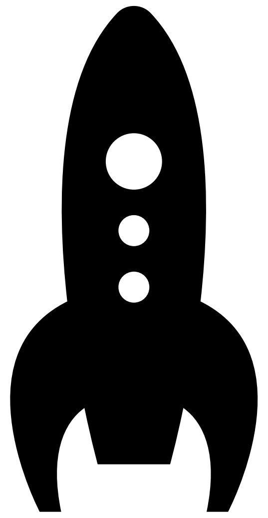


$$\text{FSS}\begin{cases}\text{Share}(f)\mapsto(\quad\quad)\\ \text{Eval}(\quad,x)+\text{Eval}(\quad,x)=f(x)\end{cases}$$

Function Secret Sharing

$$\text{FSS}\begin{cases}\text{Share}(f) \mapsto (\blacksquare \quad \blacksquare) \\ \text{Eval}(\blacksquare, x) + \text{Eval}(\blacksquare, x) = f(x)\end{cases}$$

Low end



OWF

Point functions

IT

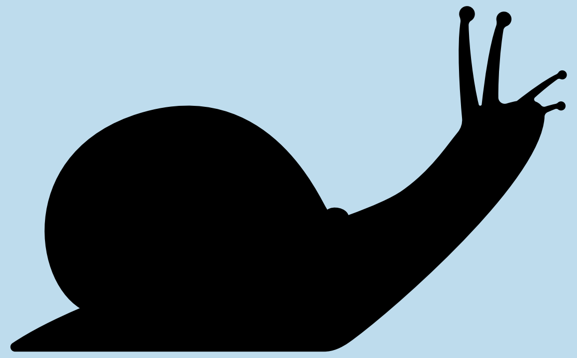
Lin. comb.

Mid end

DDH, DCR, class groups

NC^1

High end



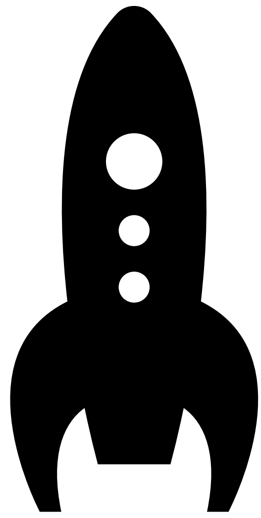
iO, Multi-key threshold FHE

All polytime functions

Function Secret Sharing

FSS
- Share(f) $\mapsto$ (🟥 🟥)
- Eval(🟥 , x) + Eval(🟥 , x) = $f(x)$

Low end



OWF — IT

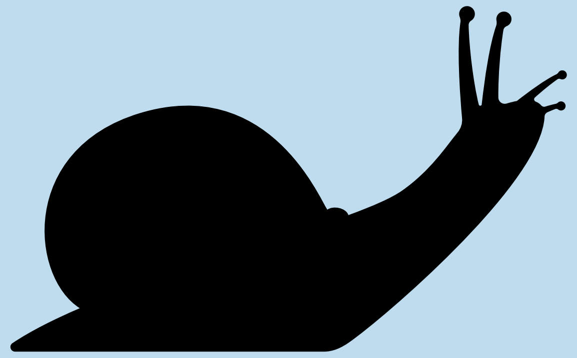
Point functions — Lin. comb.

Mid end

DDH, DCR, class groups

NC^1

High end



iO, Multi-key threshold FHE

All polytime functions

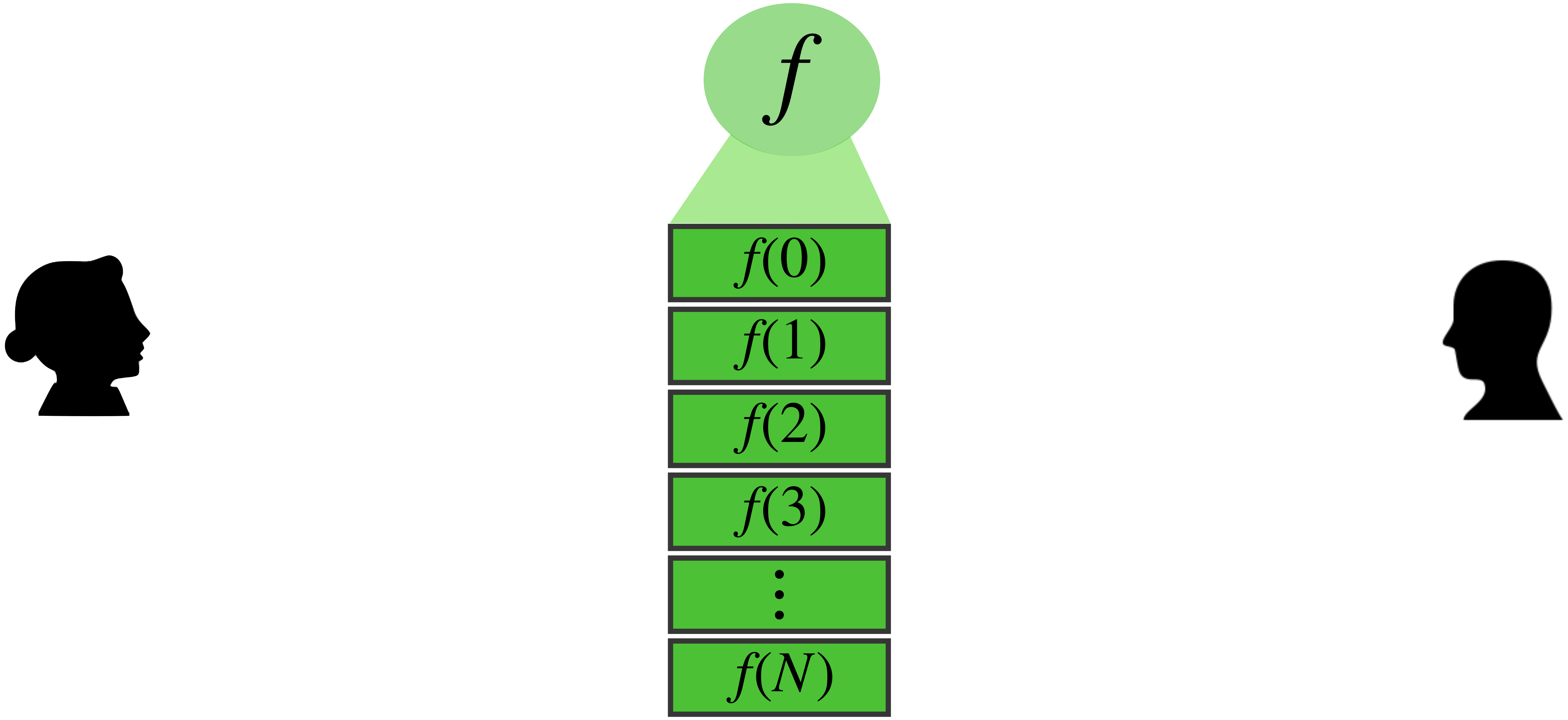
What Functions can be Shared?

Sharing an arbitrary function:

f

What Functions can be Shared?

Sharing an arbitrary function:



f

$f(0)$

$f(1)$

$f(2)$

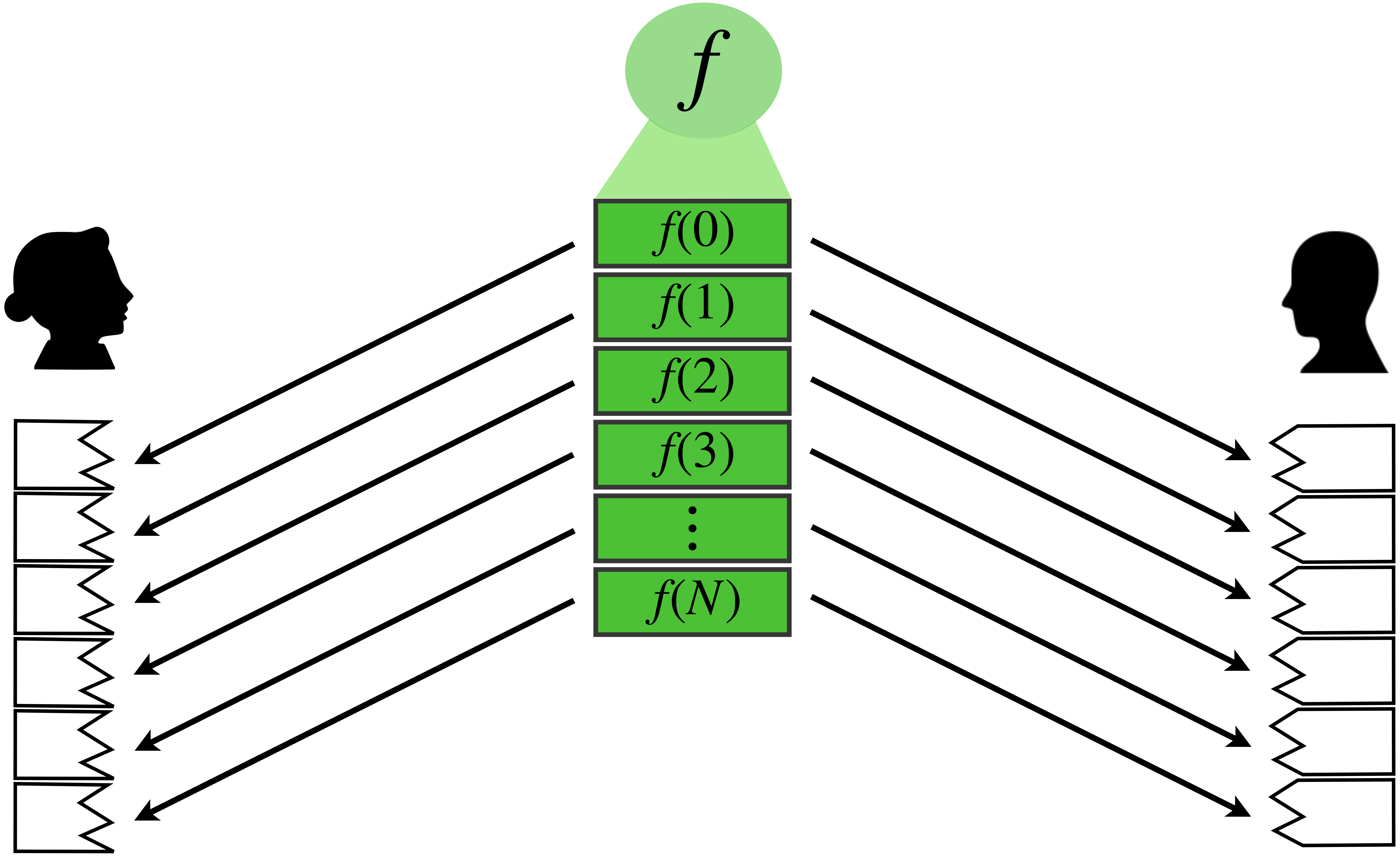
$f(3)$

$\vdots$

$f(N)$

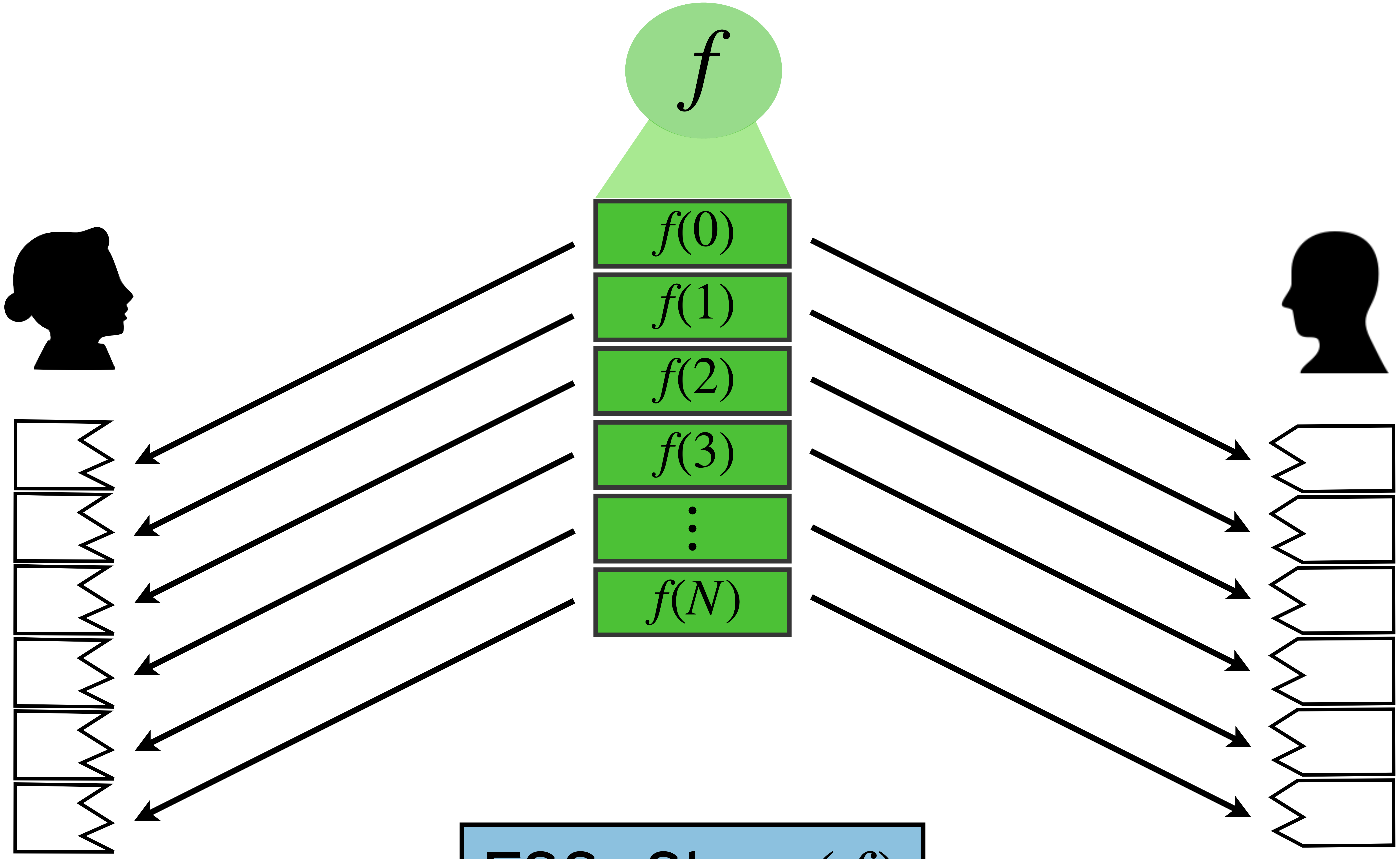
What Functions can be Shared?

Sharing an arbitrary function:



What Functions can be Shared?

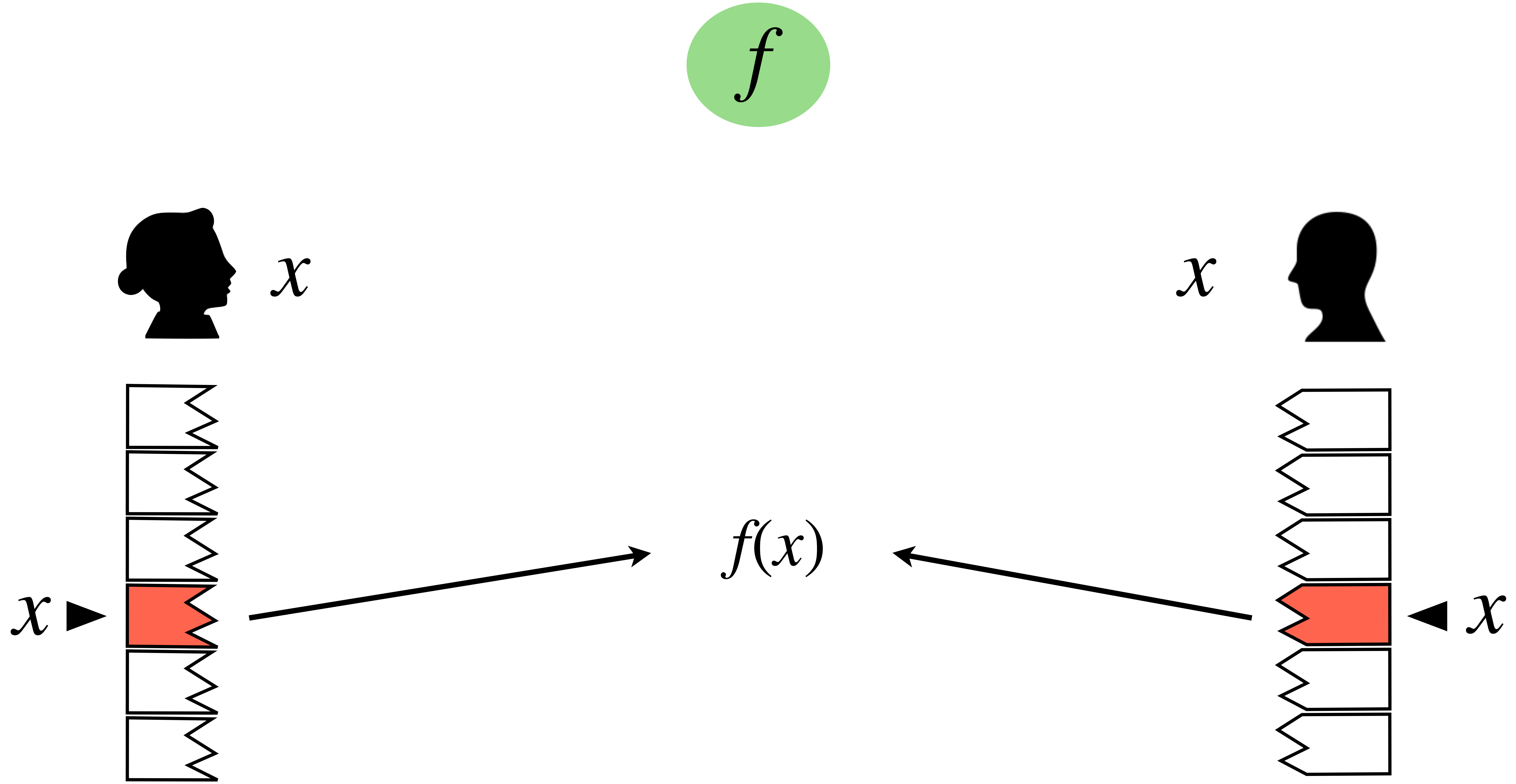
Sharing an arbitrary function:



FSS . Share(f)

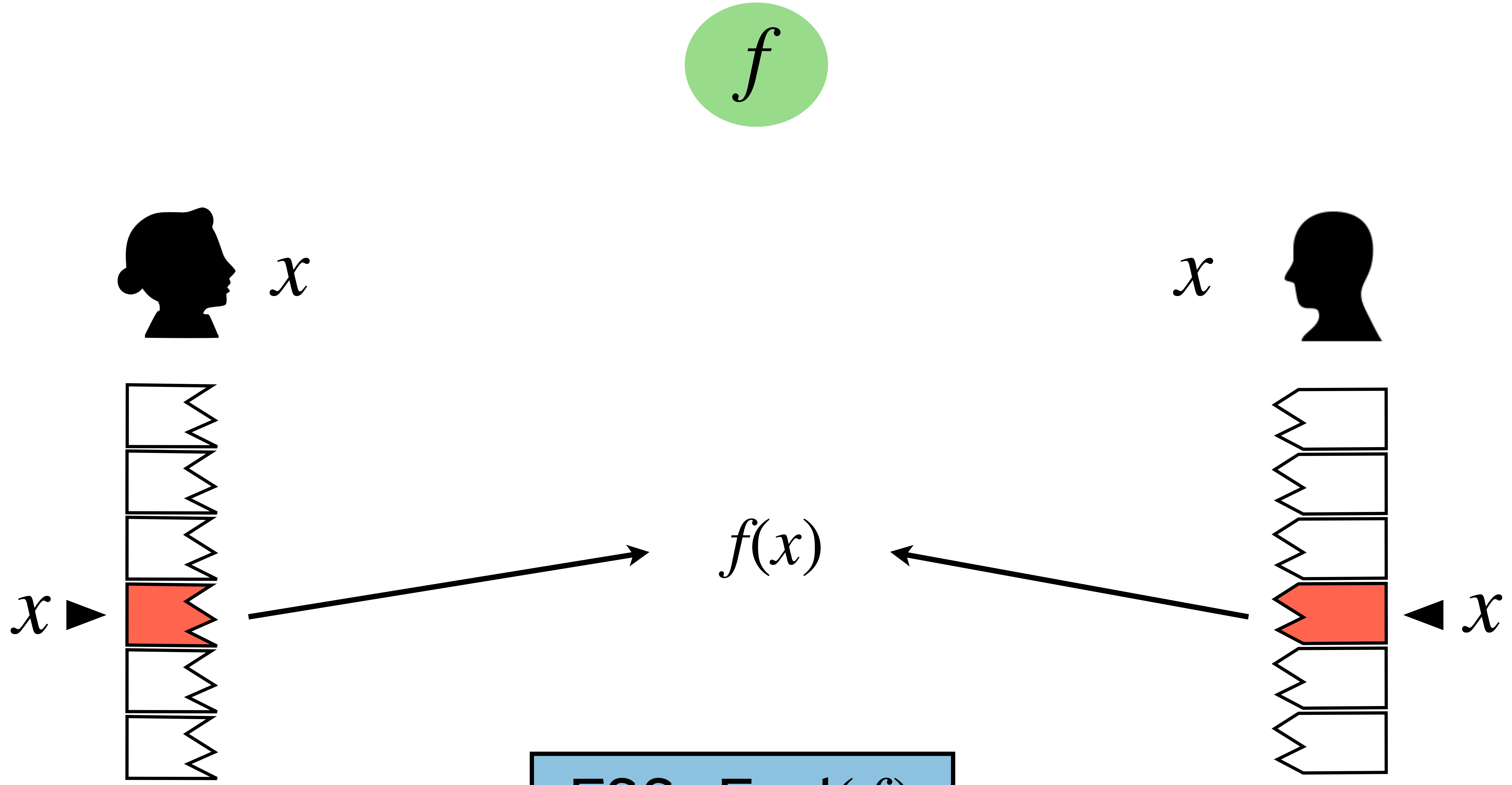
What Functions can be Shared?

Sharing an arbitrary function:



What Functions can be Shared?

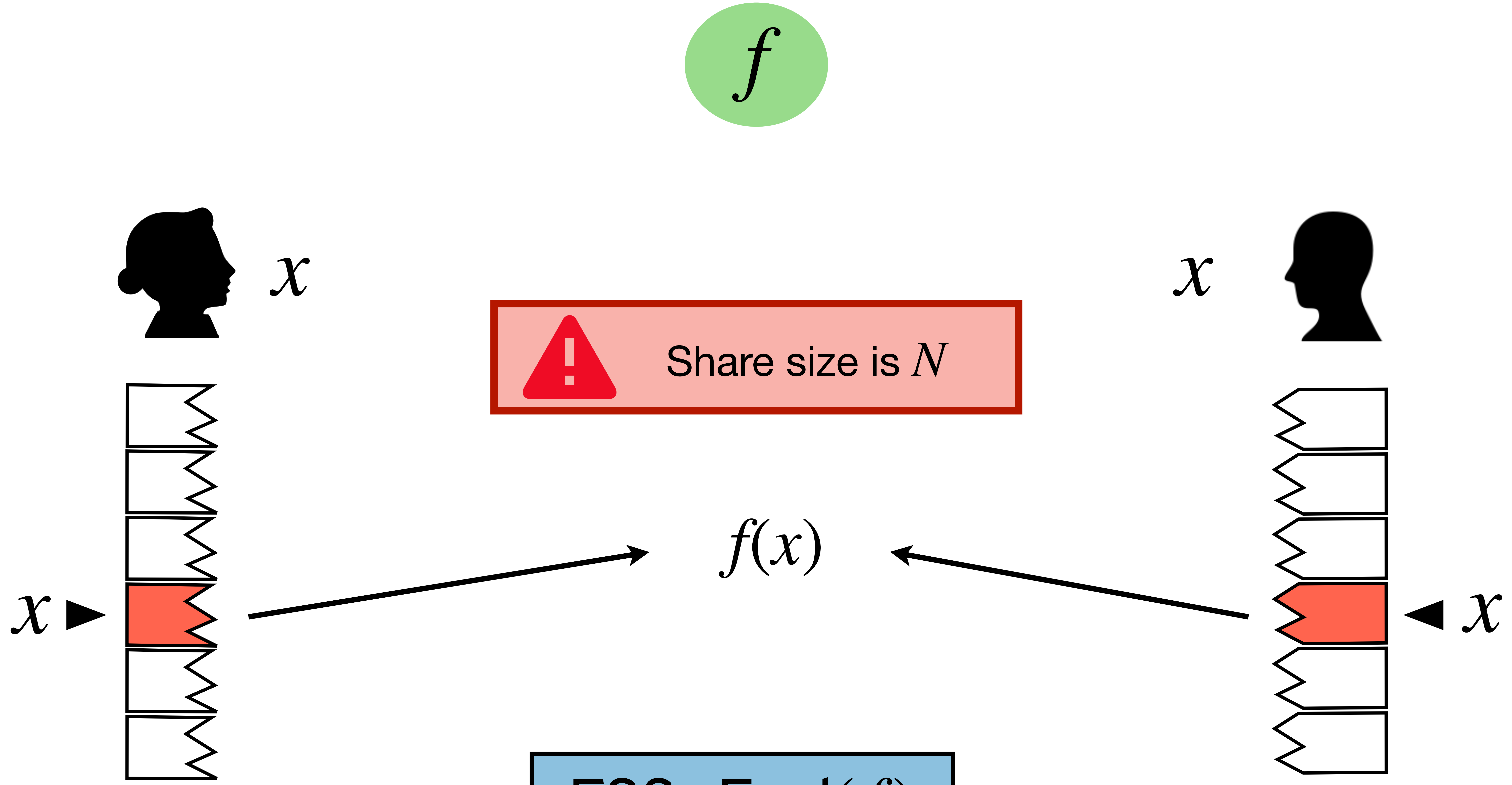
Sharing an arbitrary function:



FSS . Eval(f)

What Functions can be Shared?

Sharing an arbitrary function:



Share size is N

FSS . Eval(f)

What Functions can be Shared?

succinctly

What Functions can be Shared?

succinctly

Sharing the all zero function:

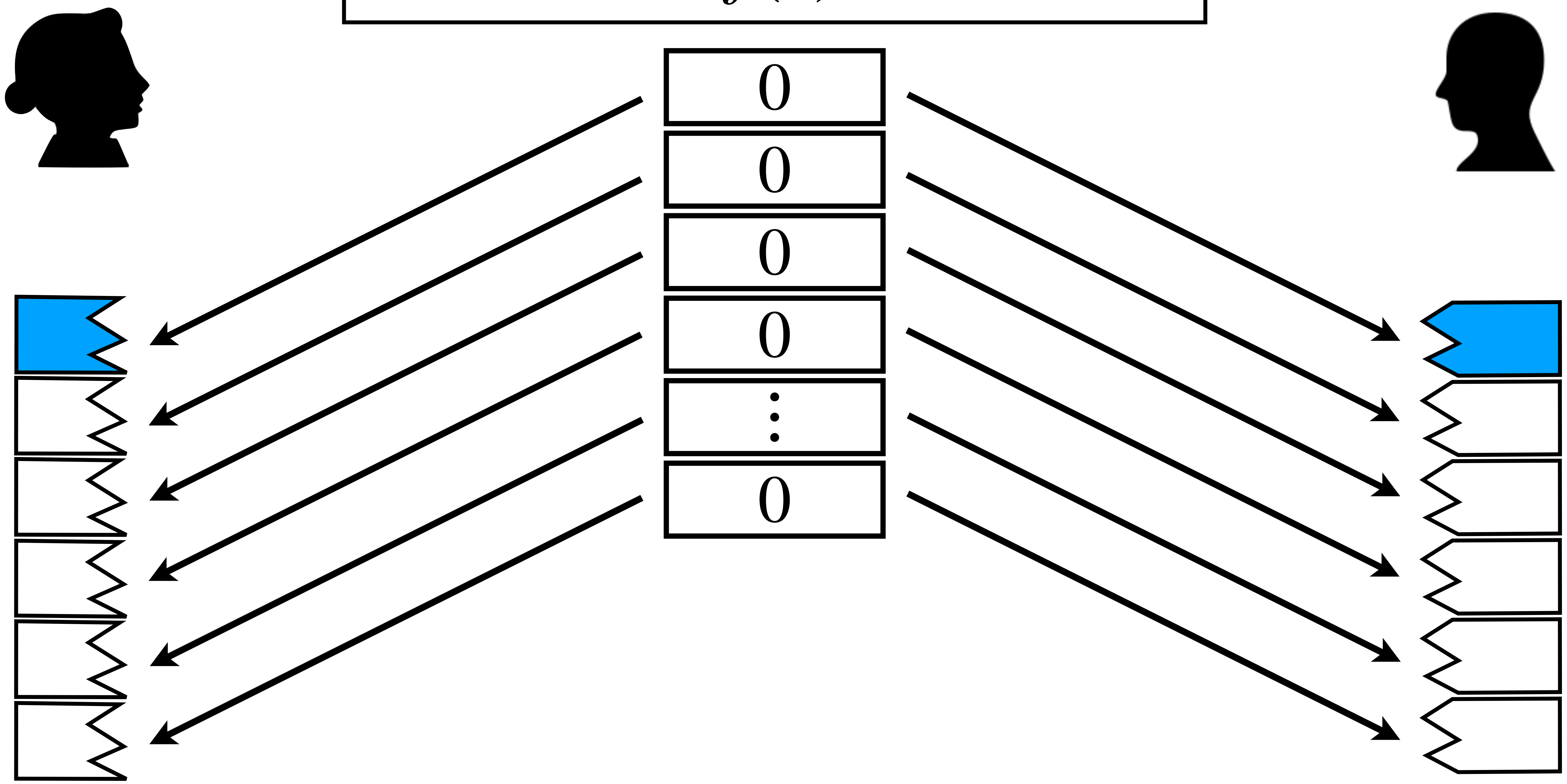
$$\forall x, f(x) = 0$$

What Functions can be Shared?

succinctly

Sharing the all zero function:

$$\forall x, f(x) = 0$$

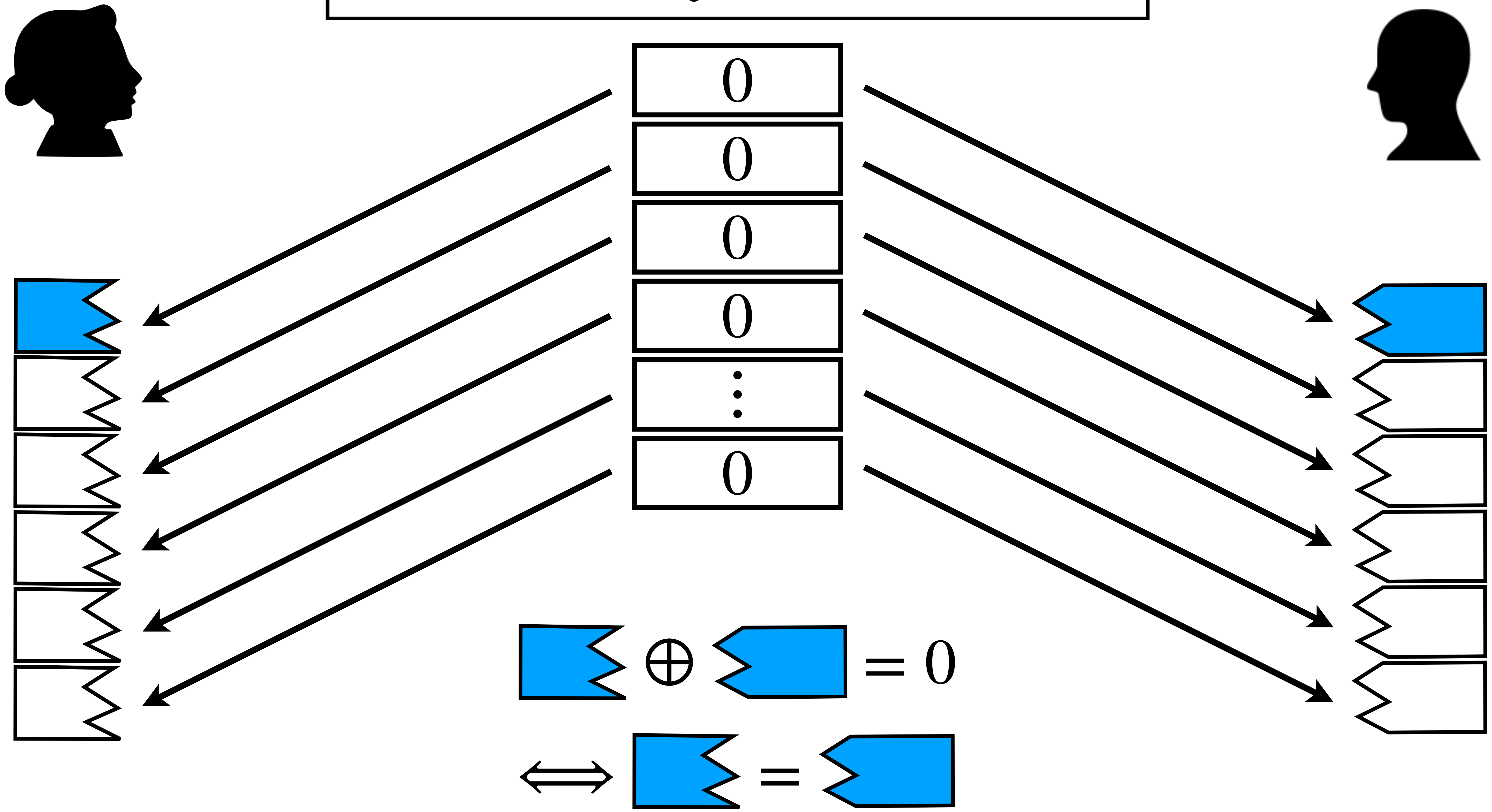


What Functions can be Shared?

succinctly

Sharing the all zero function:

$$\forall x, f(x) = 0$$

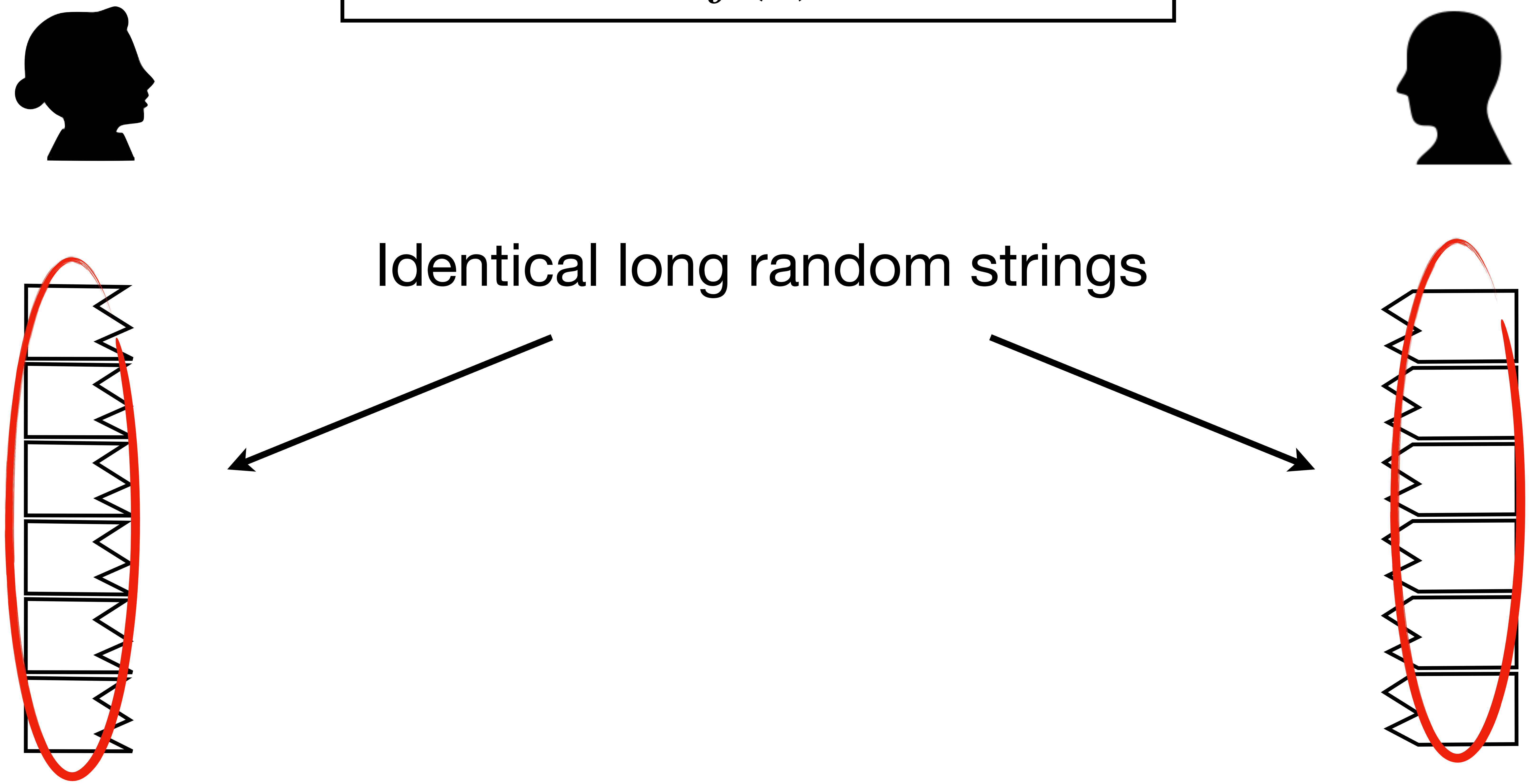


What Functions can be Shared?

succinctly

Sharing the all zero function:

$$\forall x, f(x) = 0$$



Identical long random strings

What Functions can be Shared?

succinctly

Sharing the all zero function:

$$\forall x, f(x) = 0$$

We just need a PRG!

What Functions can be Shared?

succinctly

Sharing a *point function*

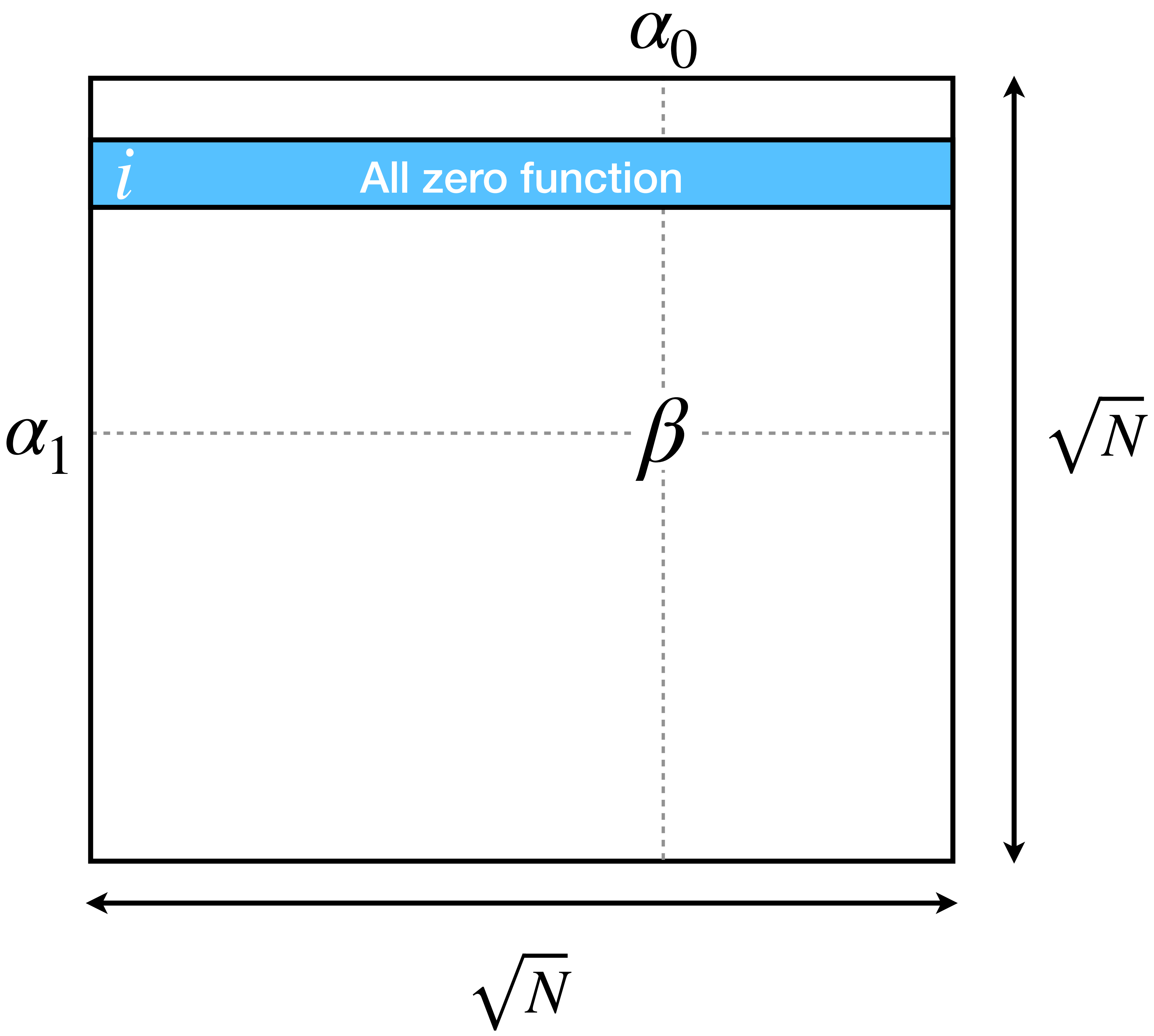
$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$

What Functions can be Shared?

succinctly

Sharing a *point function*

$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$

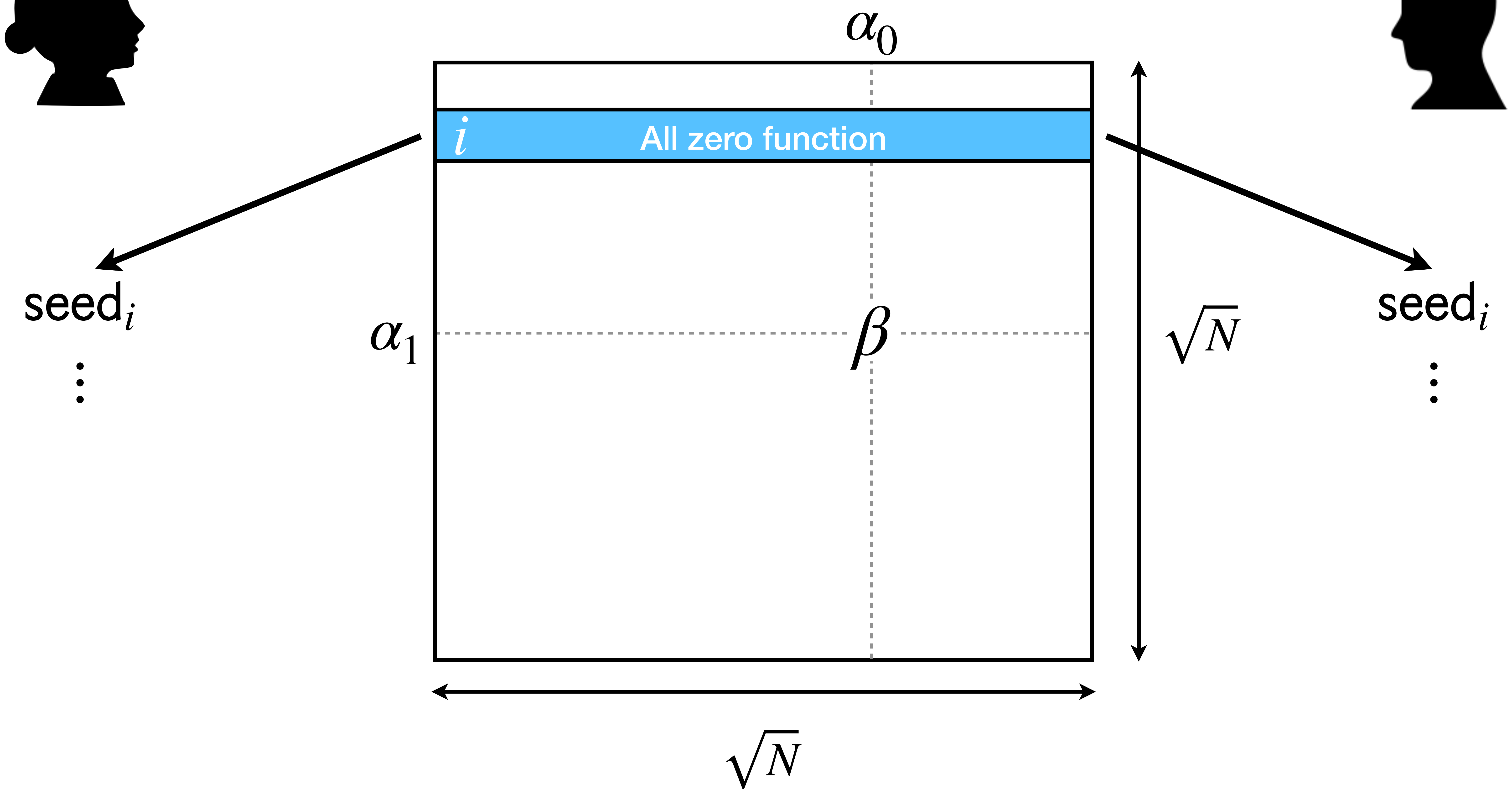


What Functions can be Shared?

succinctly

Sharing a *point function*

$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$

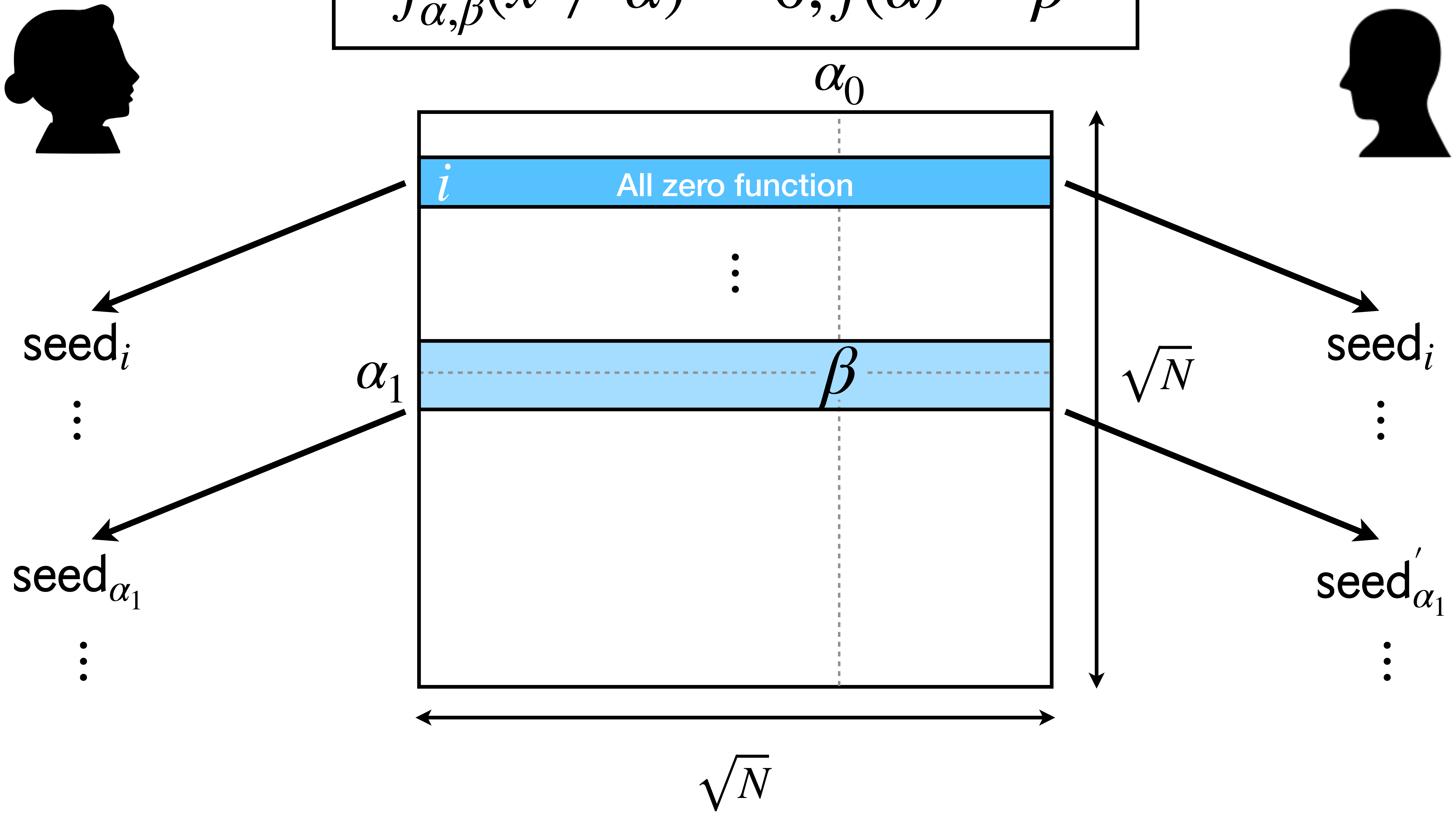


What Functions can be Shared?

succinctly

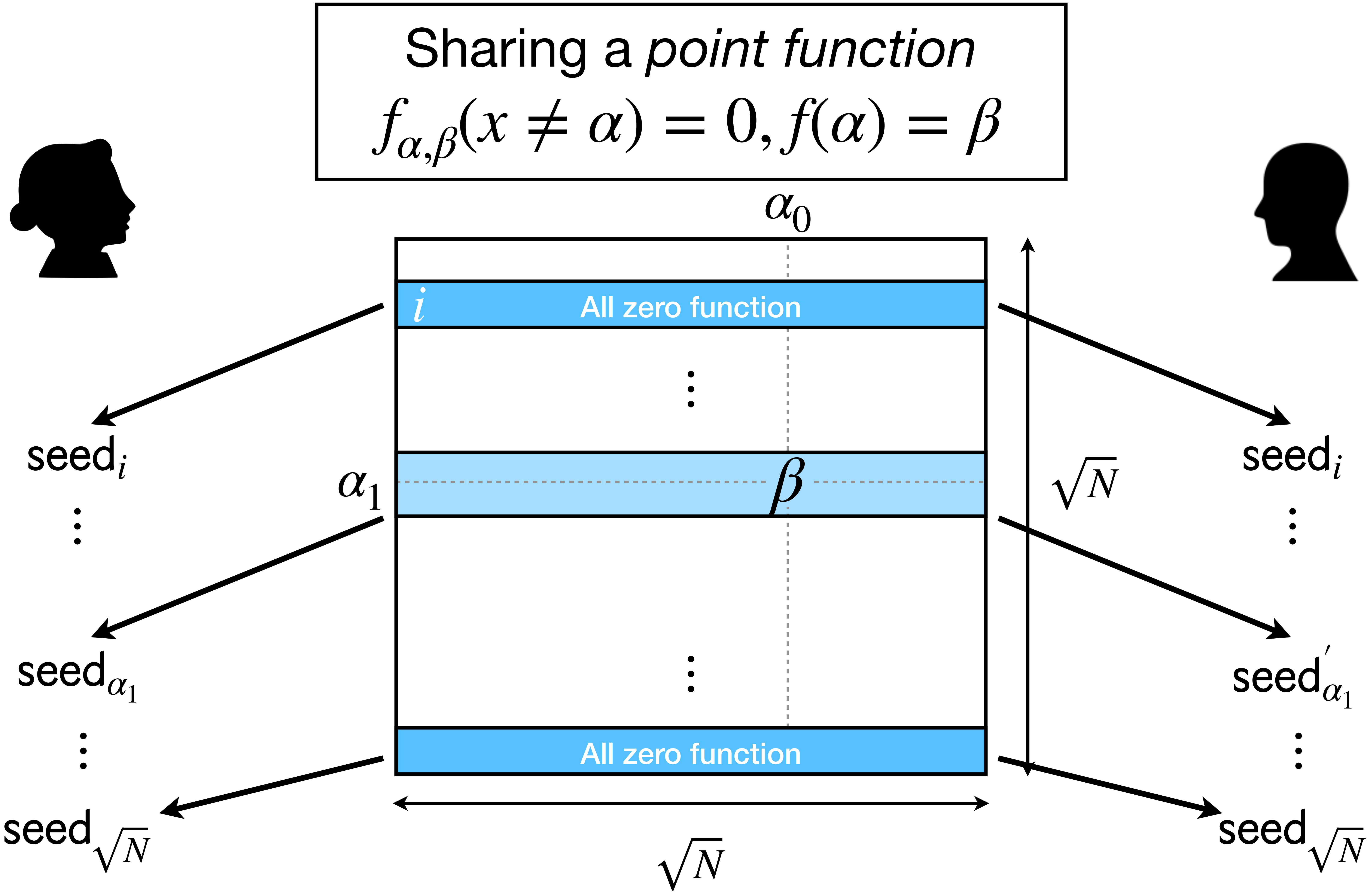
Sharing a *point function*

$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$



What Functions can be $\underset{\textbf{succinctly}}{\wedge}$ Shared?

succinctly



Sharing a *point function*

$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$

What Functions can be Shared?

succinctly

Sharing a *point function*

$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$

seed$_i$

⋮

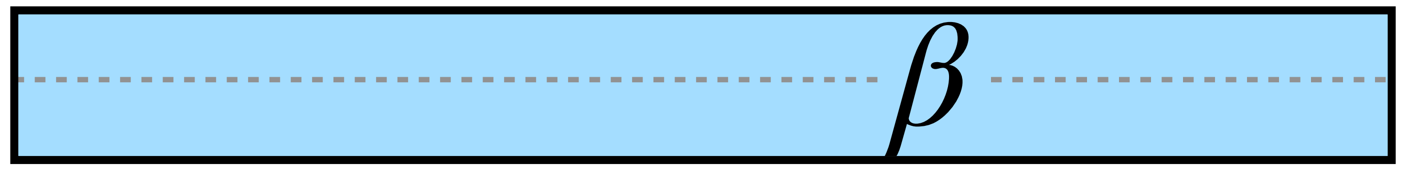
seed$_{\alpha_1}$

⋮

seed$_{\sqrt{N}}$

Public

$\Delta =$



$\oplus$ PRG(seed$_{\alpha_1}$) $\oplus$ PRG(seed$'_{\alpha_1}$)

seed$_i$

⋮

seed$'_{\alpha_1}$

⋮

seed$_{\sqrt{N}}$

What Functions can be Shared?

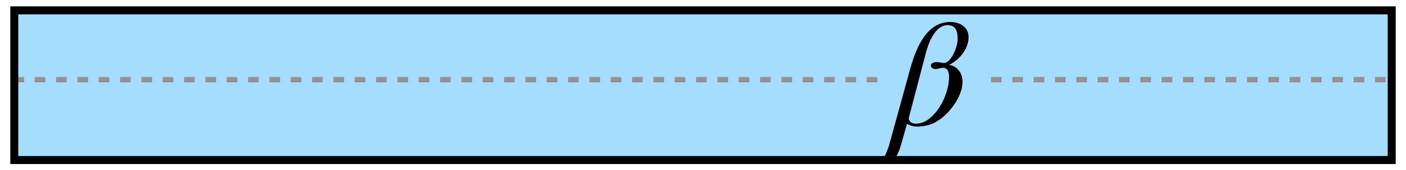
succinctly

Sharing a *point function*

$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$

Public

$\Delta =$



$\oplus \mathrm{PRG}(\mathrm{seed}_{\alpha_1}) \oplus \mathrm{PRG}(\mathrm{seed}'_{\alpha_1})$

b_i seed$_i$

$\vdots$

b_{α_1} seed$_{\alpha_1}$

$\vdots$

$b_{\sqrt{N}}$ seed$_{\sqrt{N}}$

seed$_i$ b_i

$\vdots$

seed$'_{\alpha_1}$ $1 - b_{\alpha_1}$

$\vdots$

seed$_{\sqrt{N}}$ $b_{\sqrt{N}}$

What Functions can be Shared?

succinctly

Sharing a *point function*

$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$

b_i seed$_i$

$\vdots$

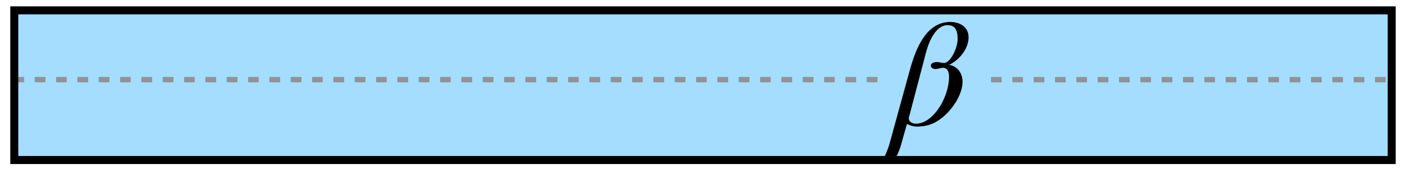
b_{α_1} seed$_{\alpha_1}$

$\vdots$

$b_{\sqrt{N}}$ seed$_{\sqrt{N}}$

Public

$\Delta =$



$\oplus$ PRG(seed$_{\alpha_1}$) $\oplus$ PRG(seed$'_{\alpha_1}$)

Eval :

PRG(seed$_j$) $\oplus \Delta \cdot b_j$

seed$_i$ b_i

$\vdots$

seed$'_{\alpha_1}$ $1 - b_{\alpha_1}$

$\vdots$

seed$_{\sqrt{N}}$ $b_{\sqrt{N}}$

What Functions can be Shared?

succinctly

Sharing a *point function*

$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$

Recursing:

b_i seed$_i$

$\vdots$

b_{α_1} seed$_{\alpha_1}$

$\vdots$

$b_{\sqrt{N}}$ seed$_{\sqrt{N}}$

Giving Δ and sharing the b_i's are both essentially sharing a $\sqrt{N}$-size point function again: we can recurse the process!

seed$_i$ b_i

$\vdots$

seed$'_{\alpha_1}$ $1 - b_{\alpha_1}$

$\vdots$

seed$_{\sqrt{N}}$ $b_{\sqrt{N}}$

What Functions can be Shared?

succinctly

Sharing a *point function*

$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$

Recursing:

b_i seed$_i$

$\vdots$

b_{α_1} seed$_{\alpha_1}$

$\vdots$

$b_{\sqrt{N}}$ seed$_{\sqrt{N}}$

Giving Δ and sharing the b_i's are both essentially sharing a $\sqrt{N}$-size point function again: we can recurse the process!

seed$_i$ b_i

$\vdots$

seed$'_{\alpha_1}$ $1 - b_{\alpha_1}$

$\vdots$

seed$_{\sqrt{N}}$ $b_{\sqrt{N}}$

This + later improvements [BGI16]:
FSS for point functions with keys of size
$O(\lambda \cdot \log N)$

Back to the PCG Template

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRG}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

We can succinctly share point functions

Back to the PCG Template

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathrm{PRG}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}}))$$

Secret sharing is additively homomorphic!

We can succinctly share point functions

Linear combinations of

Back to the PCG Template

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRG}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

Secret sharing is additively homomorphic!

We can succinctly share point functions

Linear combinations of

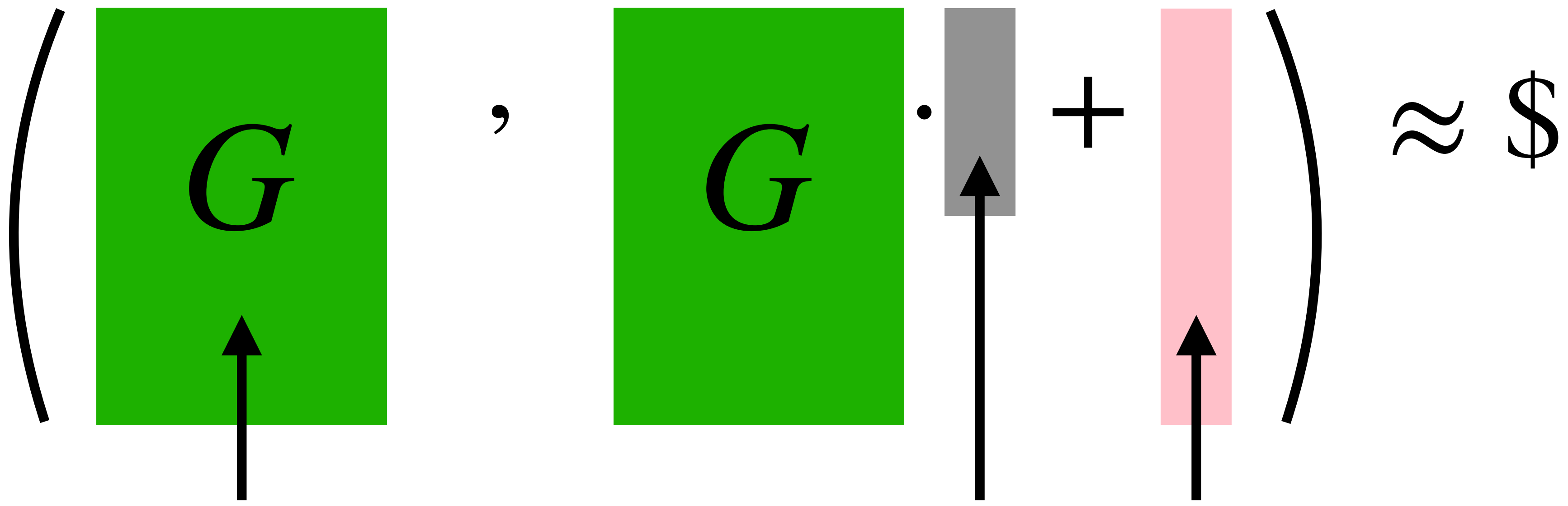
Are there any PRGs in this class?

LPN to the Rescue

The LPN assumption - primal

LPN to the Rescue

The LPN assumption - primal



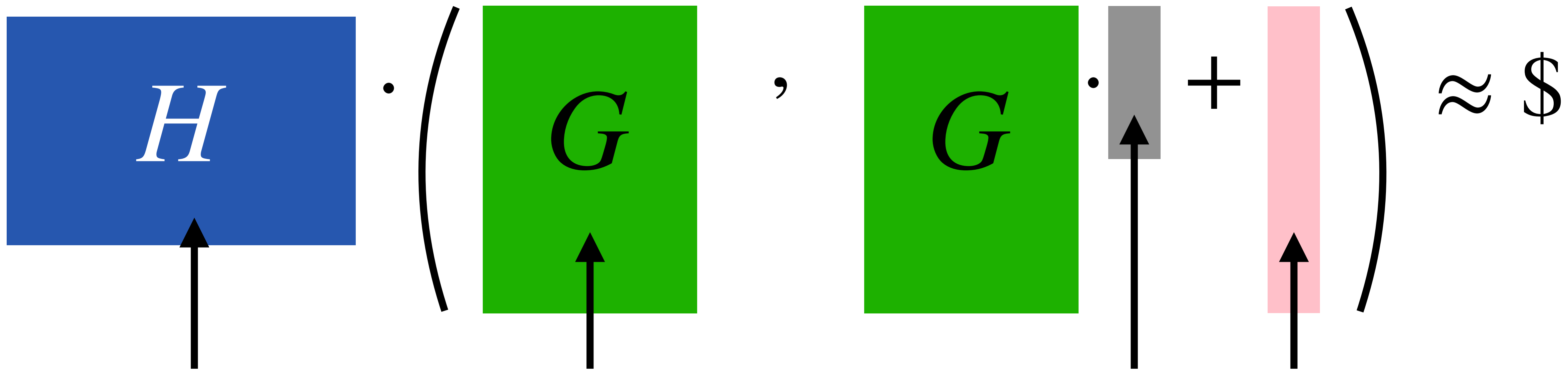
Random matrix

Short secret

Sparse noise

LPN to the Rescue

The LPN assumption - primal



Parity-check matrix of G

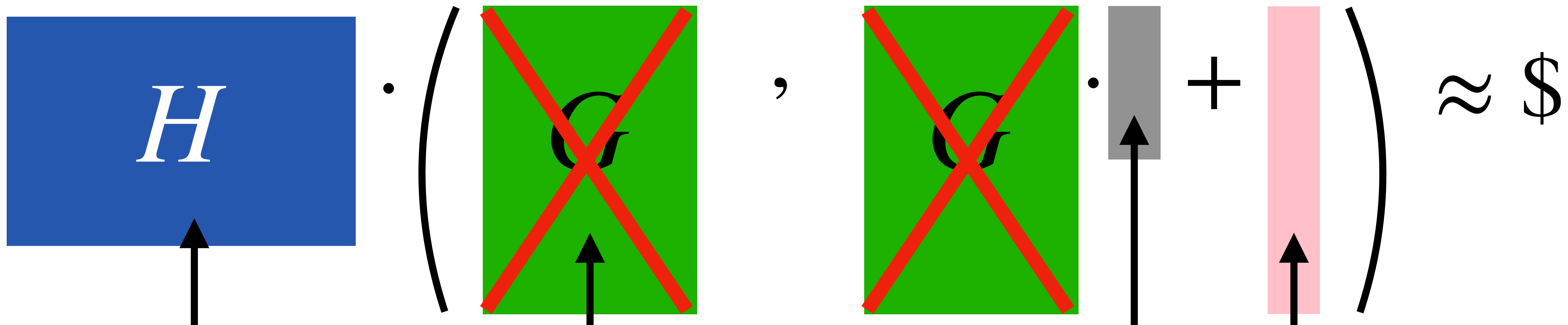
Random matrix

Short secret

Sparse noise

LPN to the Rescue

The LPN assumption - primal



Parity-check matrix of G

Random matrix

Short secret

Sparse noise

LPN to the Rescue

The LPN assumption - dual

LPN to the Rescue

The LPN assumption - dual

LPN yields a simple PRG in the class:

LPN to the Rescue

The LPN assumption - dual

Random matrix

Sparse noise

LPN yields a simple PRG in the class:

$\text{PRG} : (\alpha_i)_{i\leq t} \mapsto H \cdot \sum_{i=1}^{t} \vec{u}_{\alpha_i}$, where $\vec{u}_{\alpha_i}$ is the unit vector with a 1 at α_i

LPN to the Rescue

The LPN assumption - dual

LPN yields a simple PRG in the class:

PRG : $(\alpha_i)_{i\leq t} \mapsto \underbrace{H \cdot \sum_{i=1}^{t}}_{\text{Linear combination}} \overrightarrow{u}_{\alpha_i}$, where $\overrightarrow{u}_{\alpha_i}$ is the unit vector with a 1 at α_i

LPN to the Rescue

The LPN assumption - dual

LPN yields a simple PRG in the class:

$$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto H \cdot \underbrace{\sum_{i=1}^{t}}_{\text{Linear combination}} \underbrace{\vec{u}_{\alpha_i}}_{\text{(Truth table of) point functions}}, \text{ where } \vec{u}_{\alpha_i} \text{ is the unit vector with a 1 at } \alpha_i$$

Back to the PCG Template Again

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}\big(\overbrace{C \circ \boxed{\mathrm{PRG}}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}})\big)$$

We have FSS for a class that contains a PRG

Back to the PCG Template Again

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathrm{PRG}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}}))$$

👍 We have FSS for a class that contains a PRG

The heavy lifting in the many subsequent works boils down to:

- Making the PRG more efficient
- Adding support for more complex C

Back to the PCG Template Again

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRG}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

👍 We have FSS for a class that contains a PRG

The heavy lifting in the many subsequent works boils down to:

- Making the PRG more efficient
- Adding support for more complex C

} Both questions are deeply rooted in (combinatorial and algebraic) coding theory

Back to the PCG Template Again

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathrm{PRG}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}}))$$

We have FSS for a class that contains a PRG

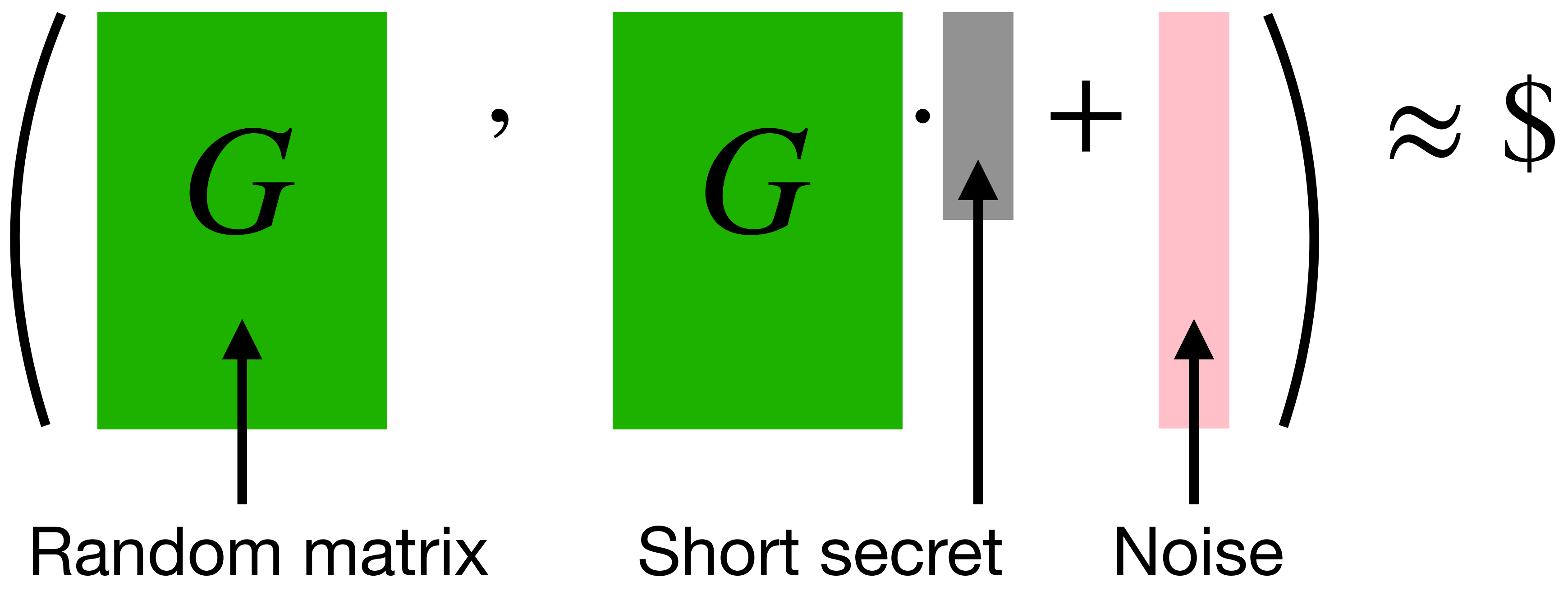
The heavy lifting in the many subsequent works boils down to:

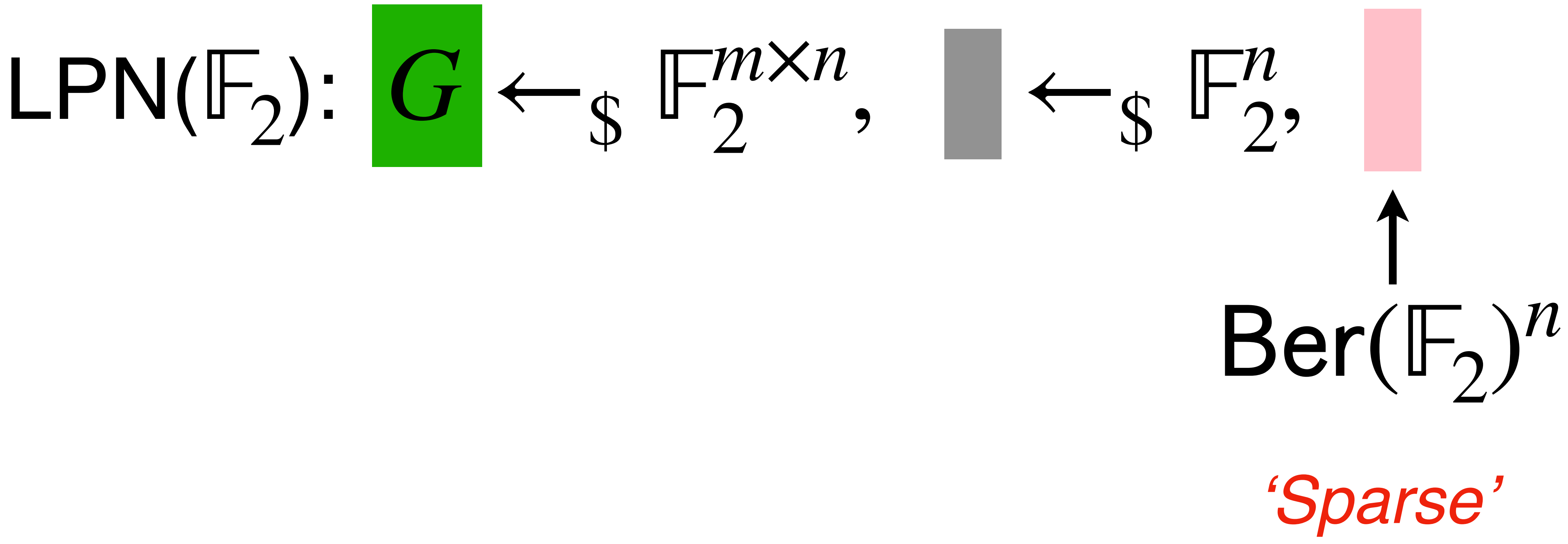
- Making the PRG more efficient
- Adding support for more complex C

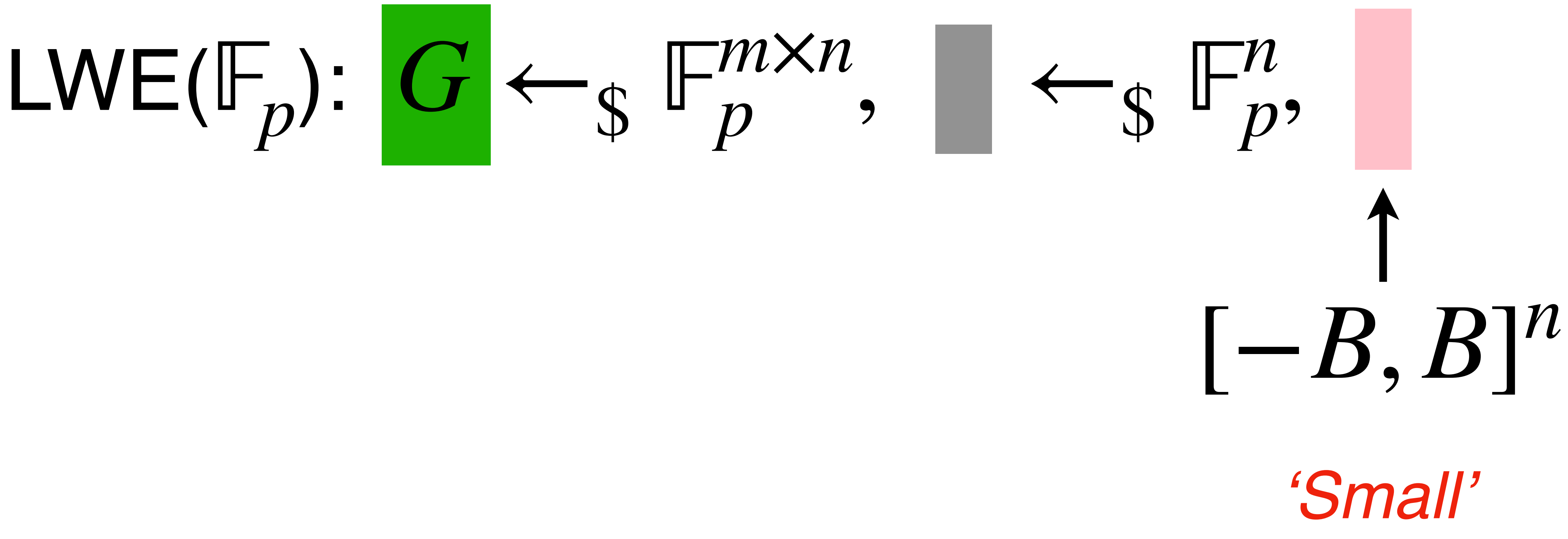
$\}$ Both questions are deeply rooted in (combinatorial and algebraic) coding theory

Digression: LPN versus LWE

LPN and LWE

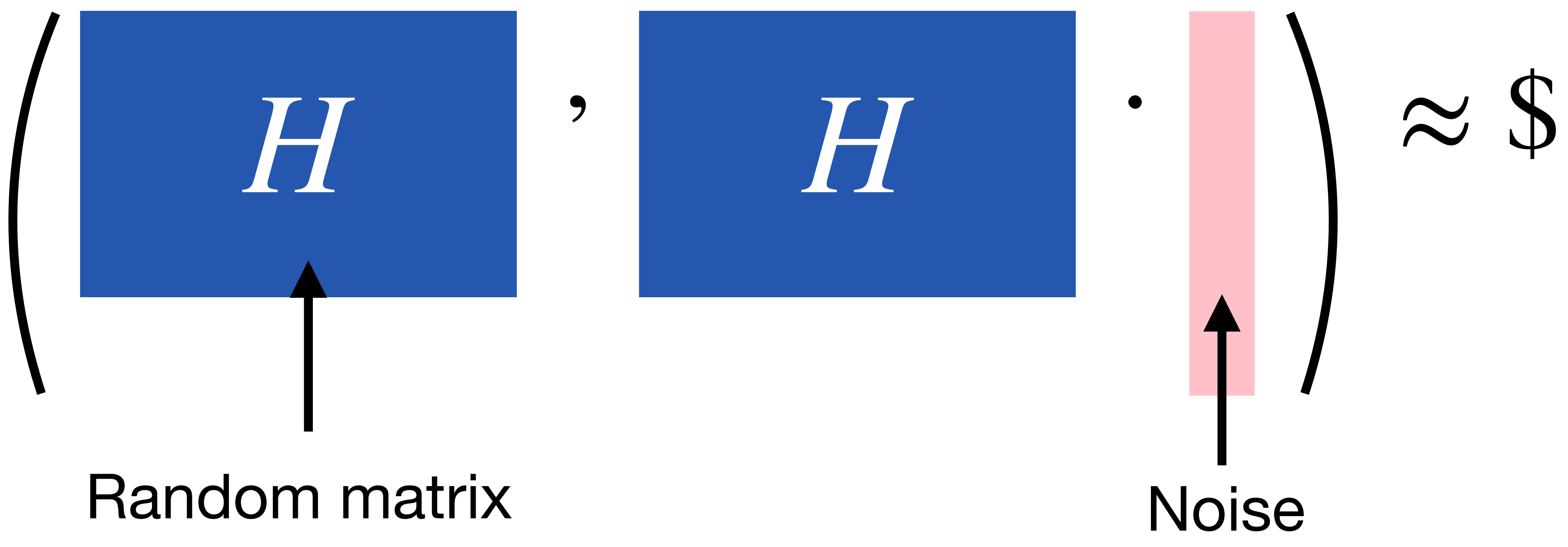


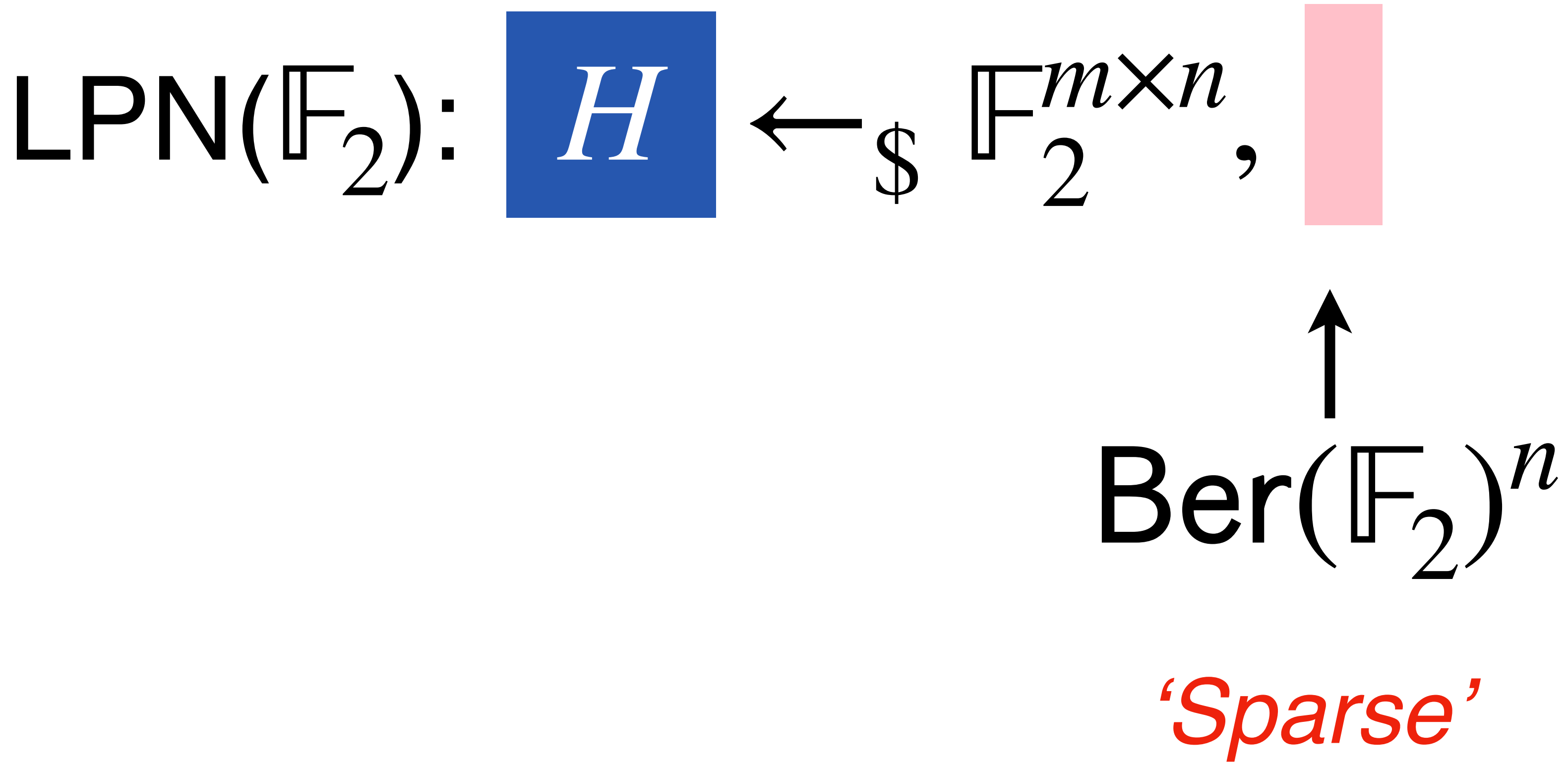




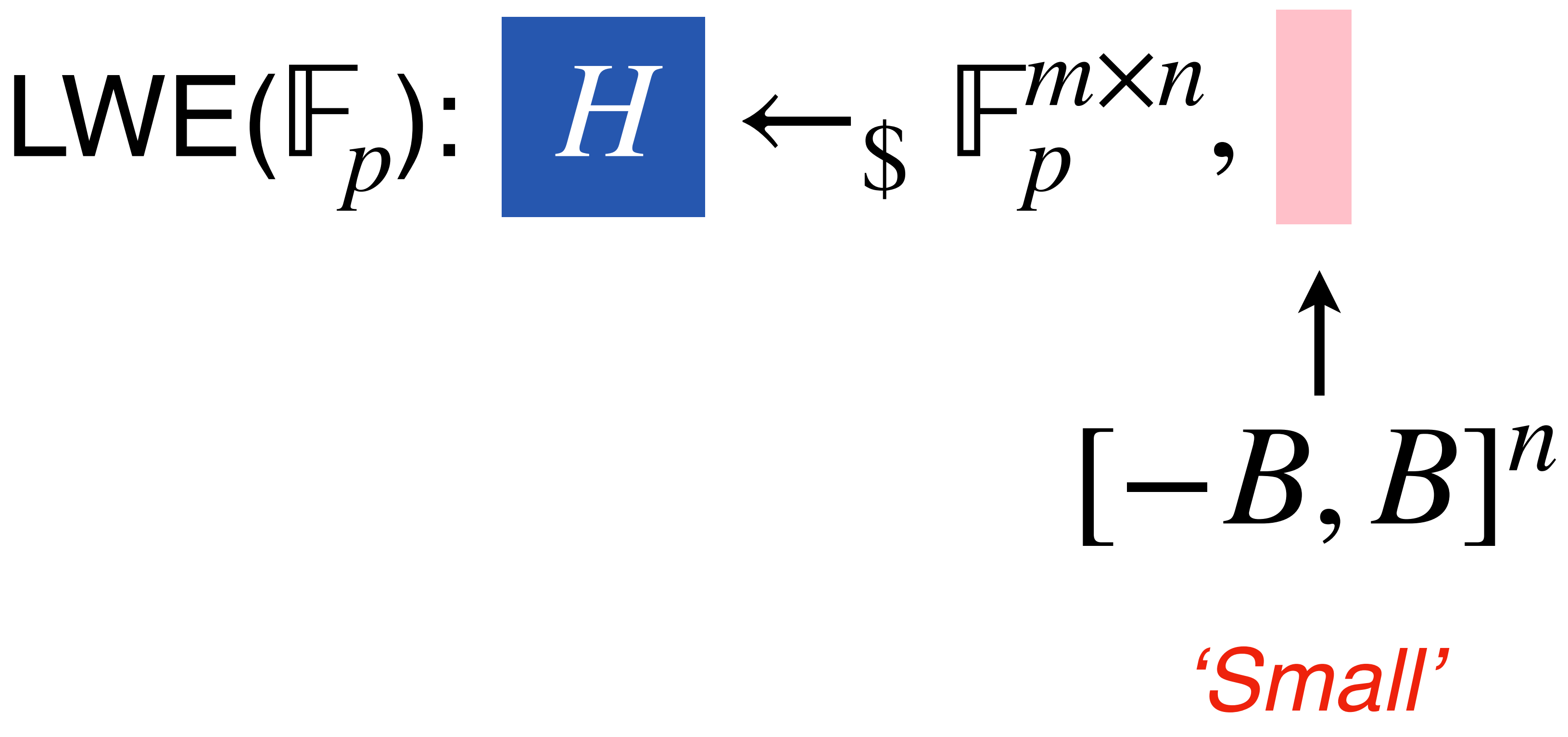
Digression: LPN versus LWE

LPN and LWE





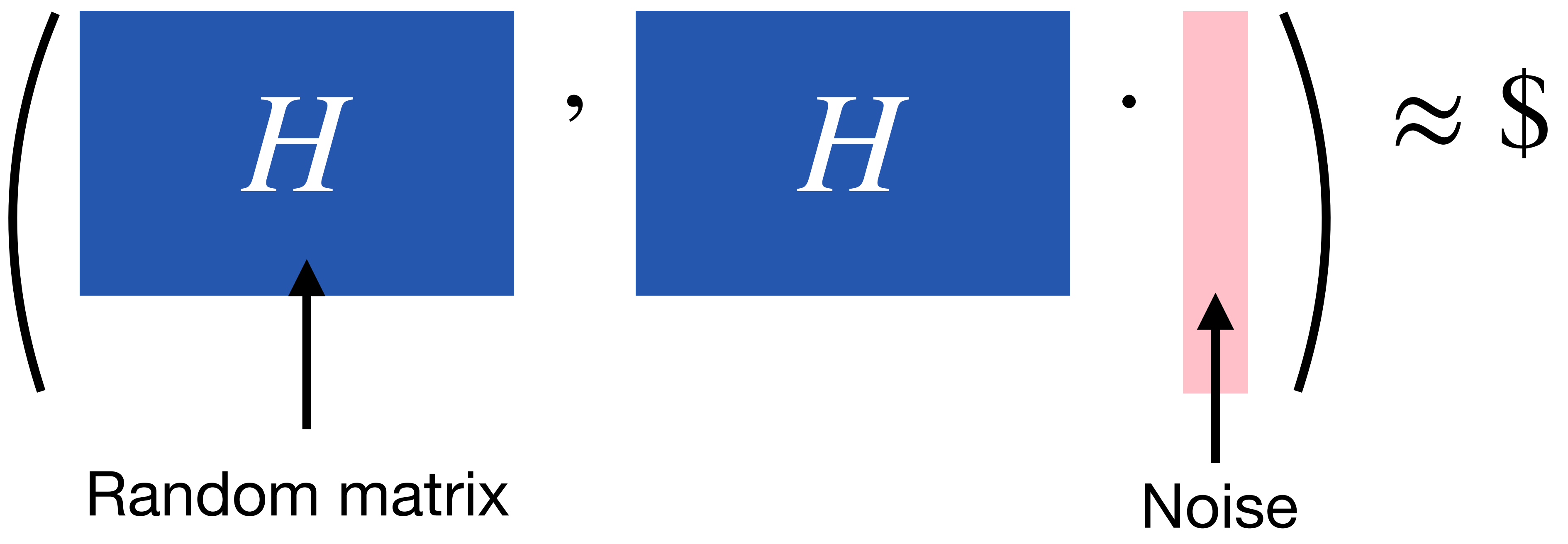
'Sparse'



'Small'

Digression: LPN versus LWE

LPN and LWE



LPN($\mathbb{F}_2$): $H \leftarrow_\$ \mathbb{F}_2^{m\times n}$,

Ber($\mathbb{F}_2$)^n

'Sparse'

Statistical security

LWE($\mathbb{F}_p$): $H \leftarrow_\$ \mathbb{F}_p^{m\times n}$,

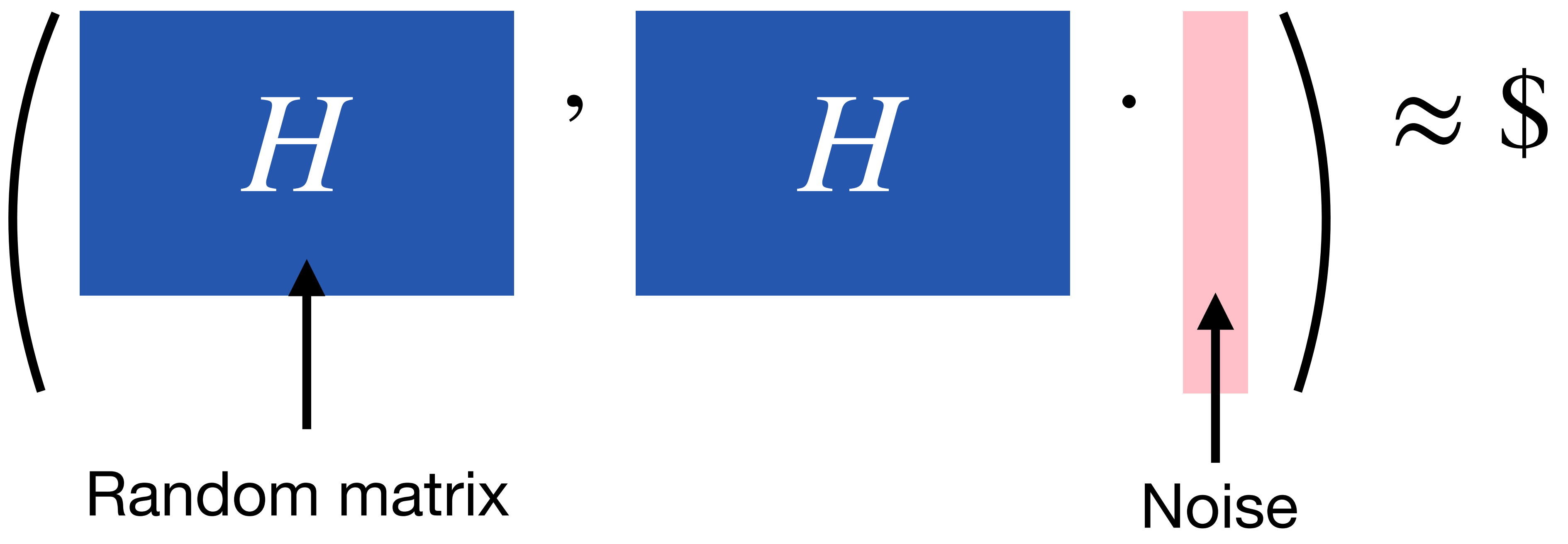
$[-B, B]^n$

'Small'

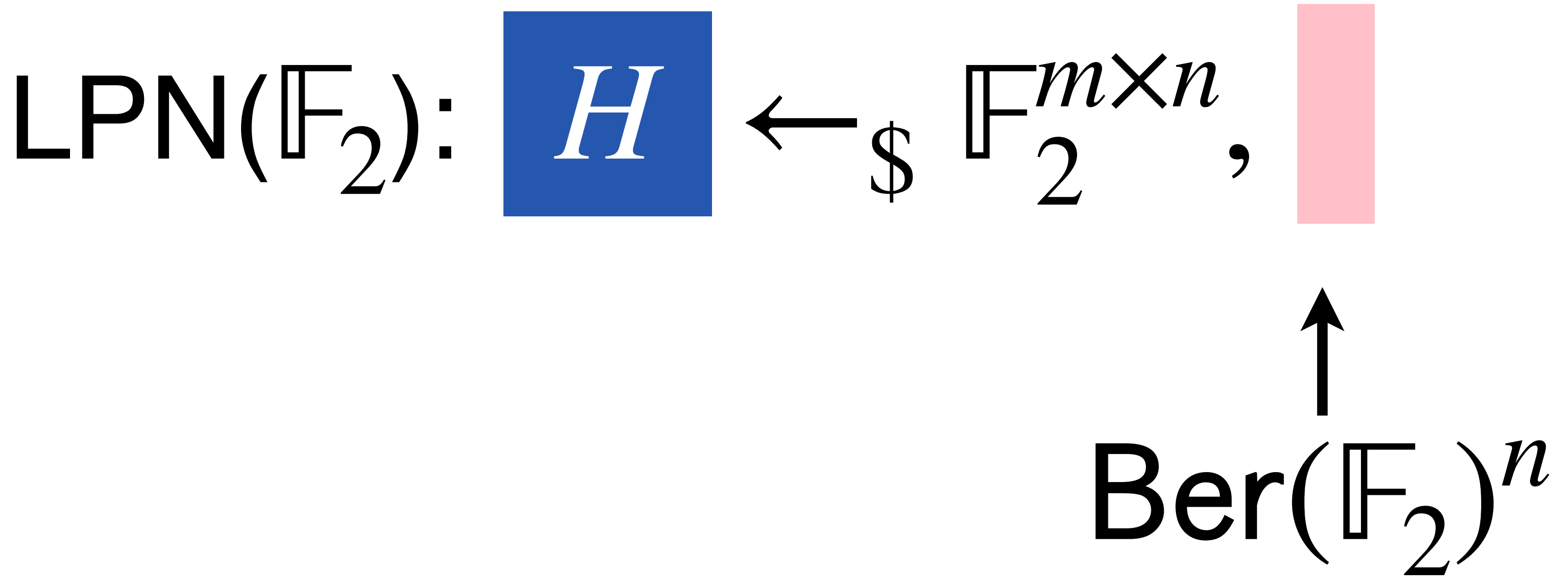
$O(n)$ entropy in the noise $\Longrightarrow$ LHL, statistical security, lattice trapdoors, lossiness…

Digression: LPN versus LWE

LPN and LWE



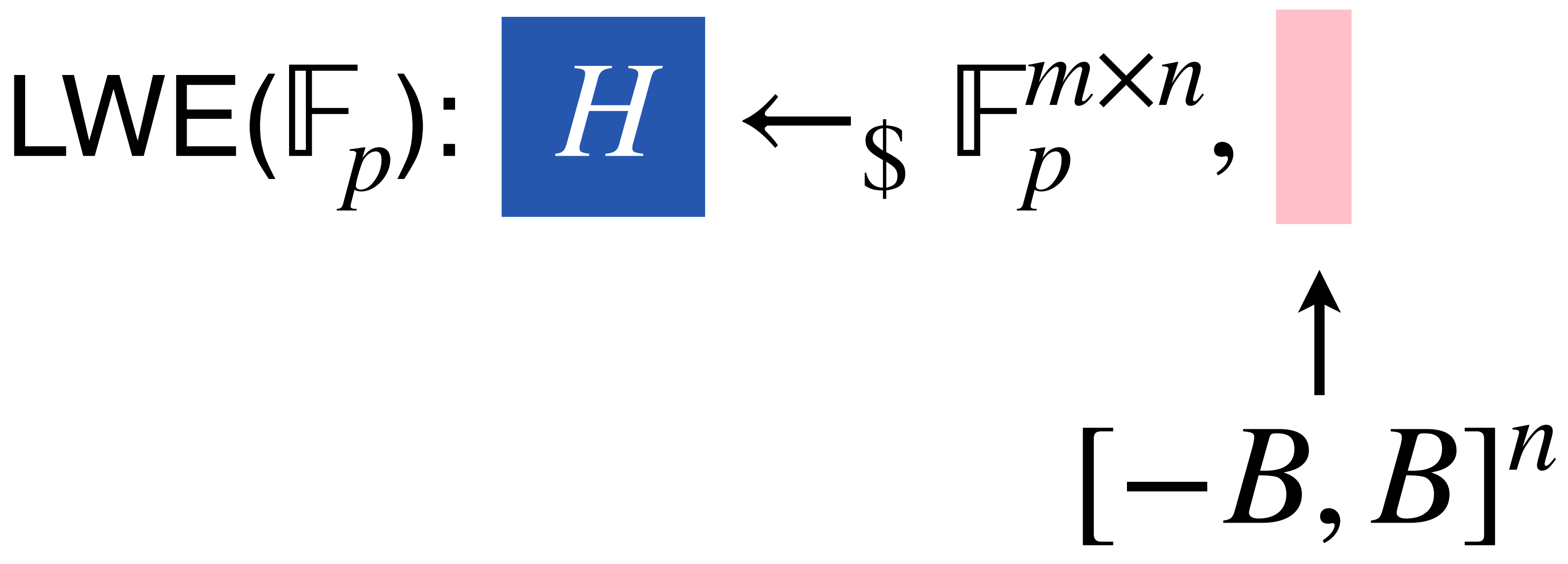
Compressibility



'Sparse'

$t \cdot \log n \ll n$ entropy in the noise $\Longrightarrow$ compressibility! Crucially used in recent results: PCGs, but also iO and batch OT.

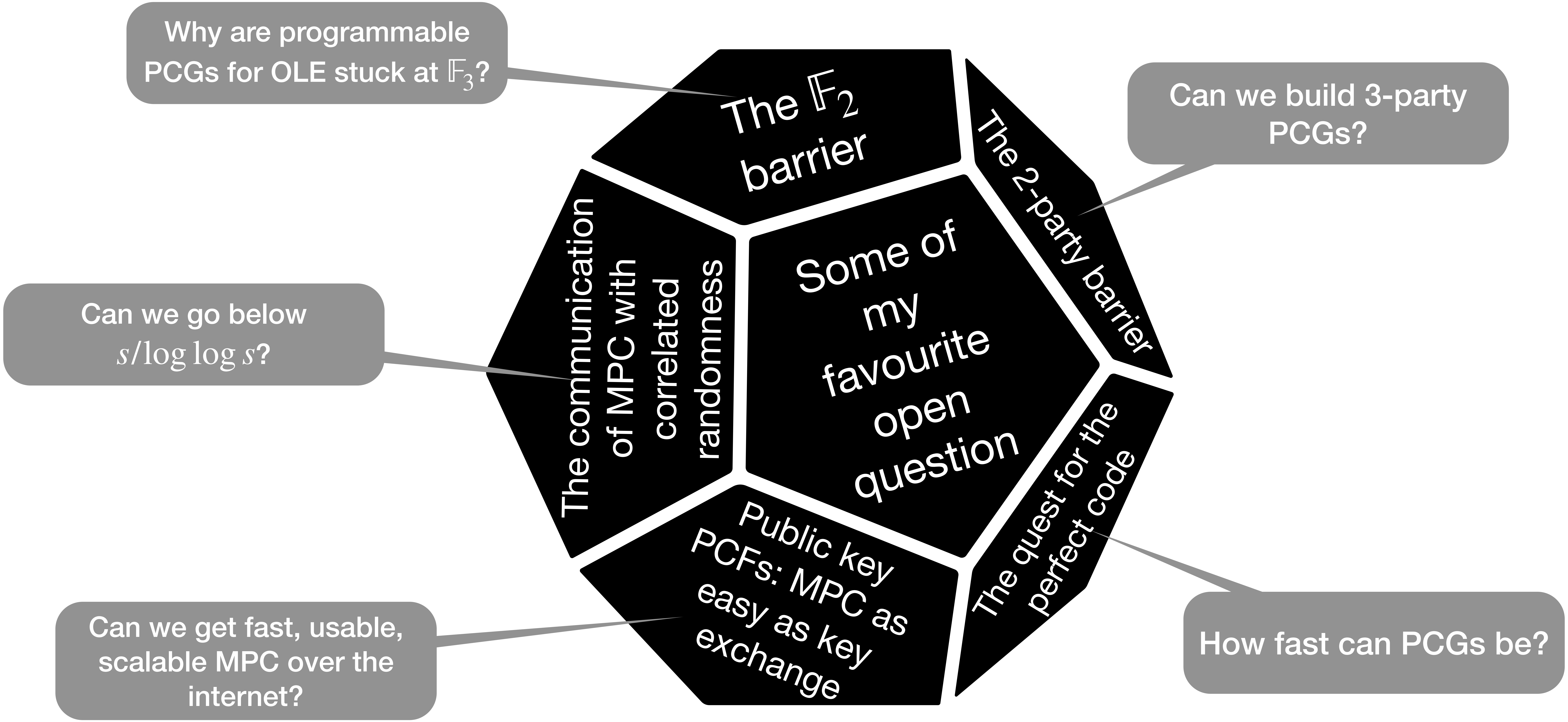
Statistical security



'Small'

$O(n)$ entropy in the noise $\Longrightarrow$ LHL, statistical security, lattice trapdoors, lossiness…

Some of my Favourite Open Questions



Some of my favourite open question

The $\mathbb{F}_2$ barrier

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The 2-party barrier

Can we build 3-party PCGs?

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

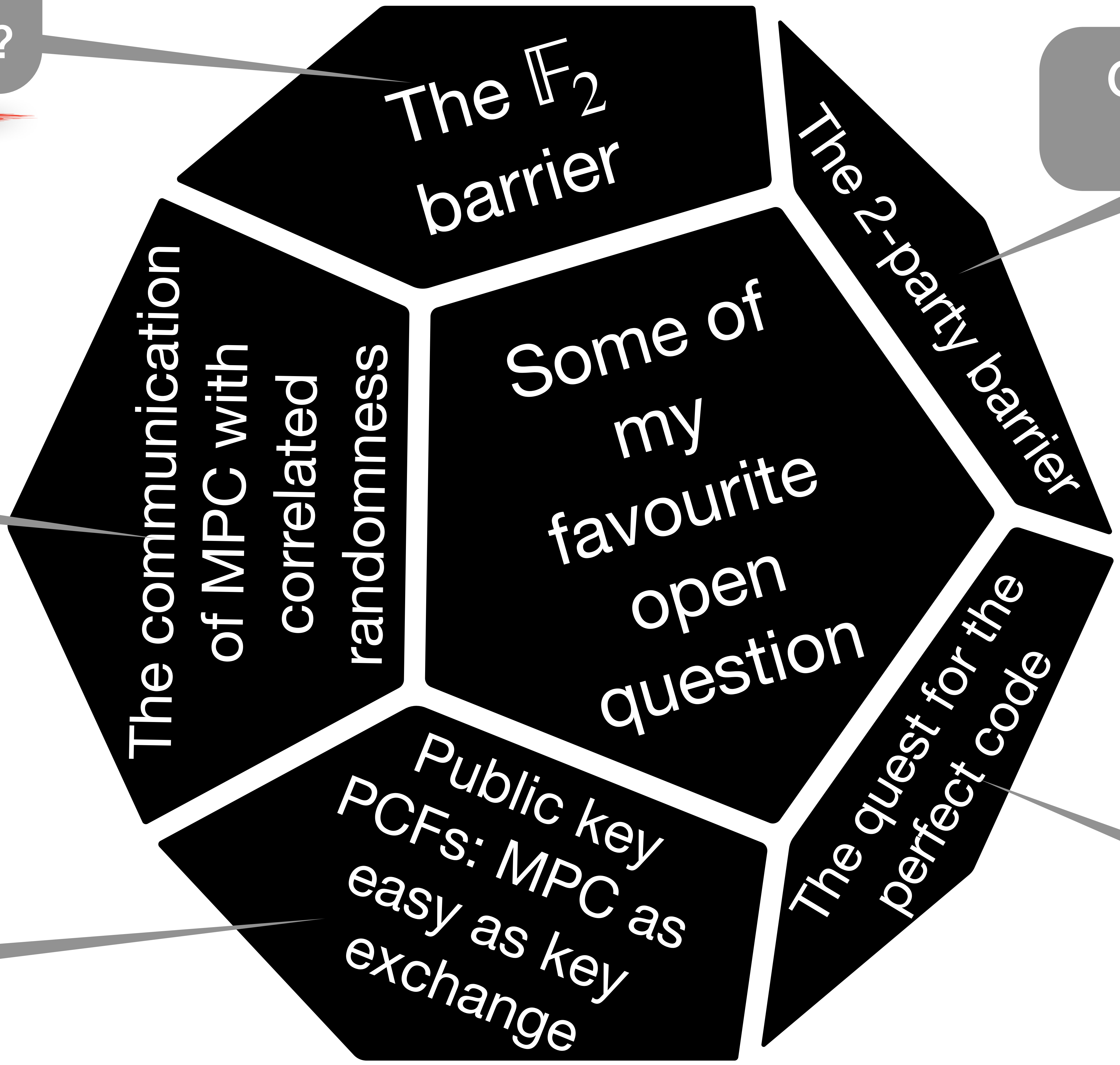
Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The quest for the perfect code

How fast can PCGs be?

Some of my Favourite Open Questions



Some of my favourite open question

The $\mathbb{F}_2$ barrier

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The 2-party barrier

Can we build 3-party PCGs?

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

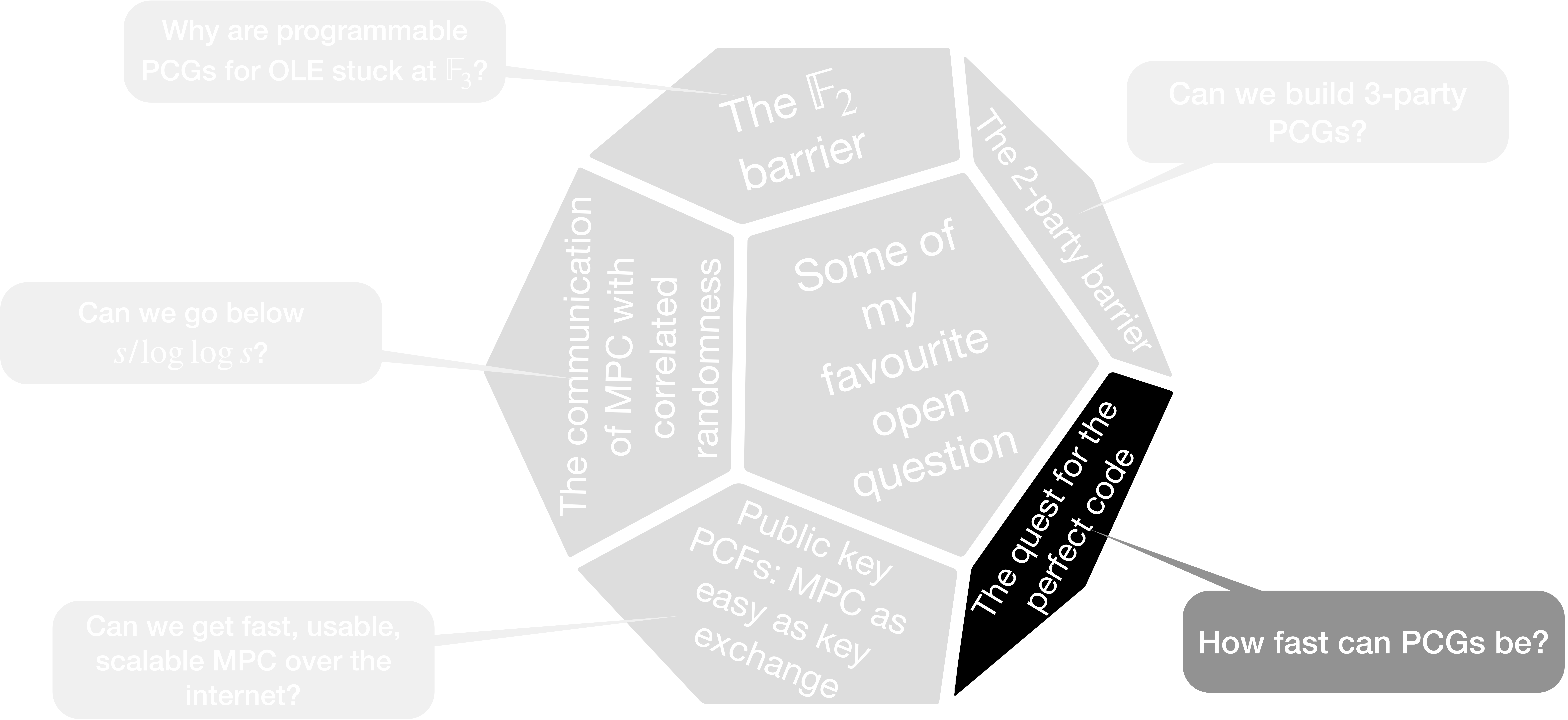
Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The quest for the perfect code

How fast can PCGs be?

Some of my Favourite Open Questions



Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The $\mathbb{F}_2$ barrier

The 2-party barrier

Can we build 3-party PCGs?

Can we go below $s/\log\log s$?

The communication of MPC with correlated randomness

Some of my favourite open question

The quest for the perfect code

How fast can PCGs be?

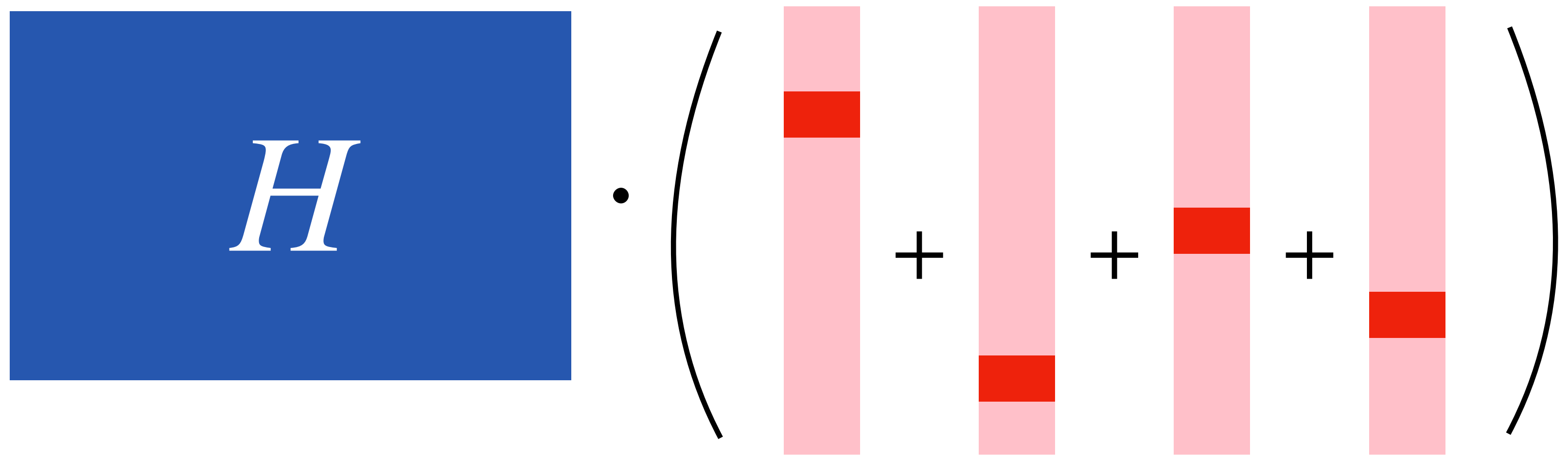
Can we get fast, usable, scalable MPC over the internet?

Public key PCFs: MPC as easy as key exchange

Making the PRG more Efficient

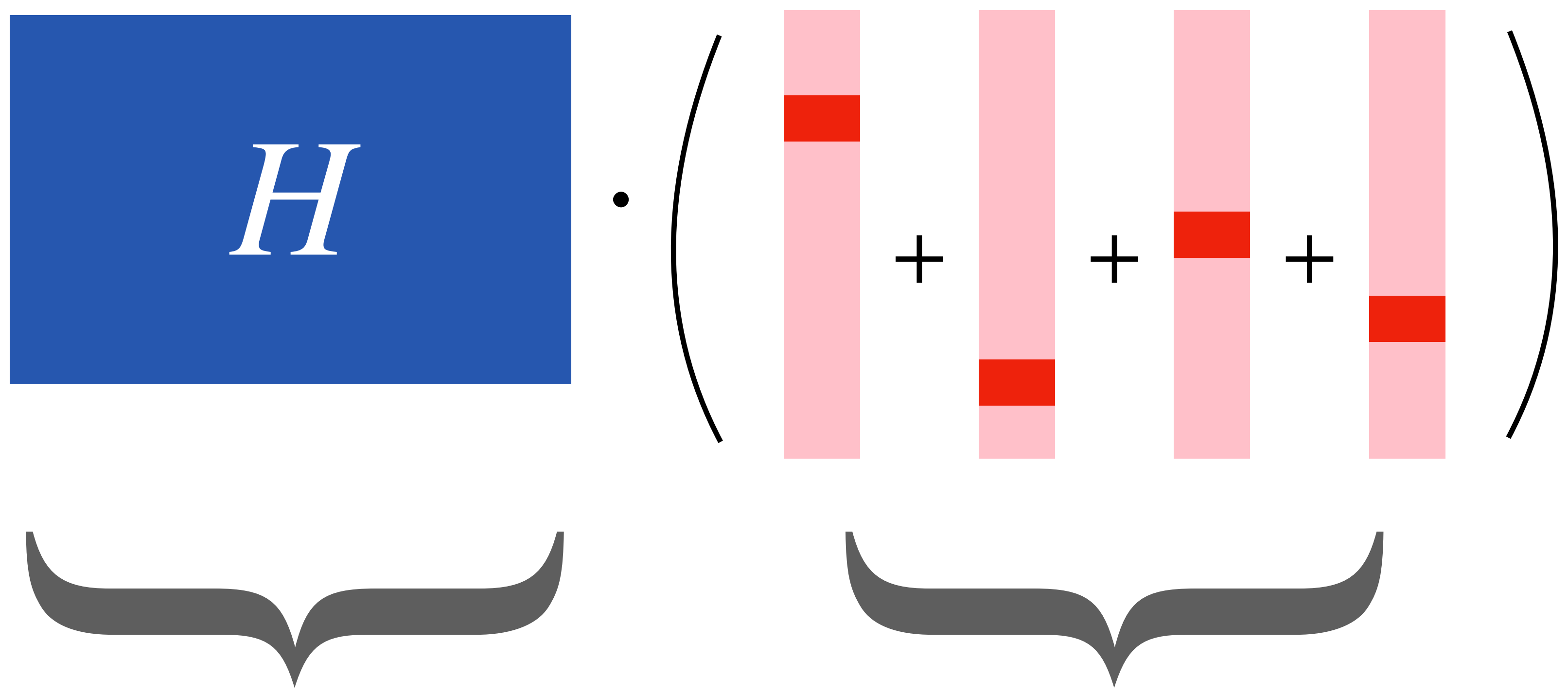
Making the PRG more Efficient

$\mathsf{PRG} : (\alpha_i)_{i\le t} \mapsto$



Making the PRG more Efficient

$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto$

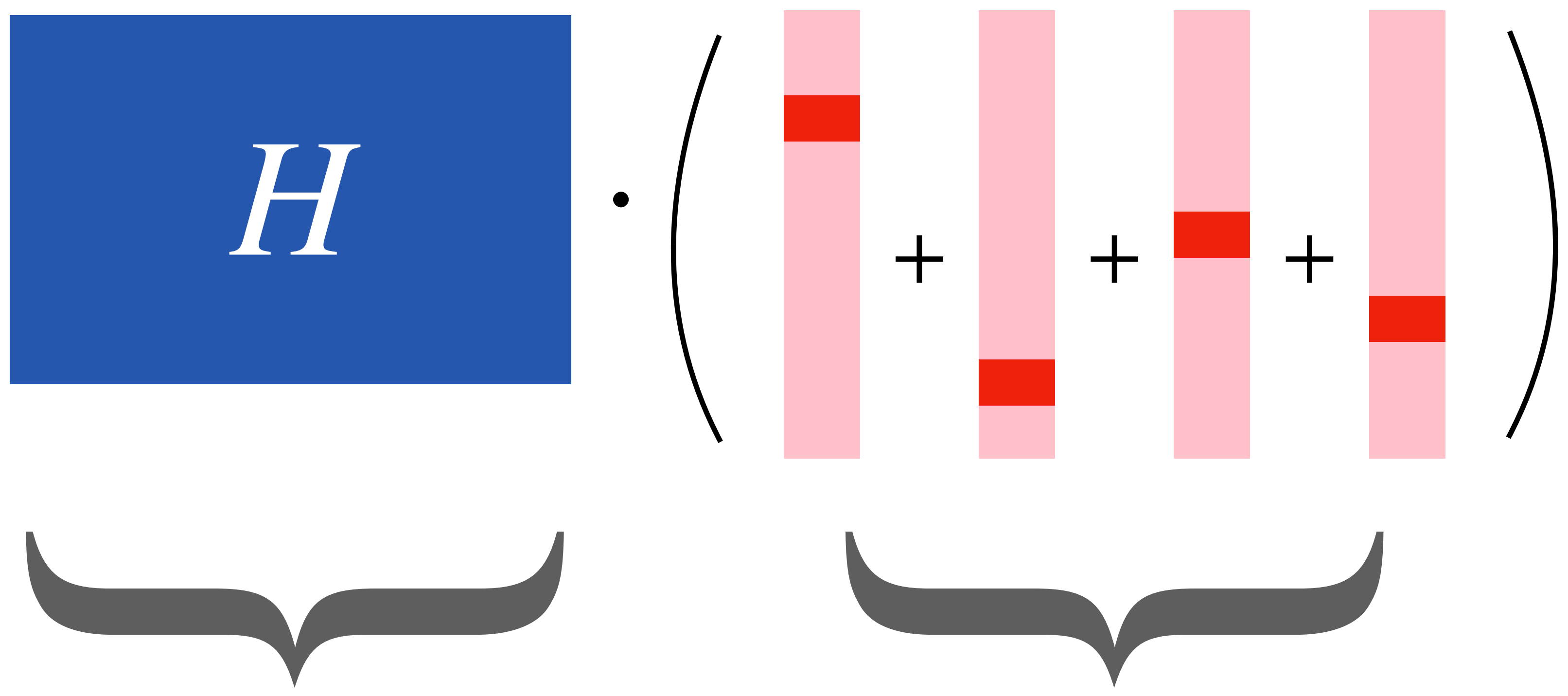


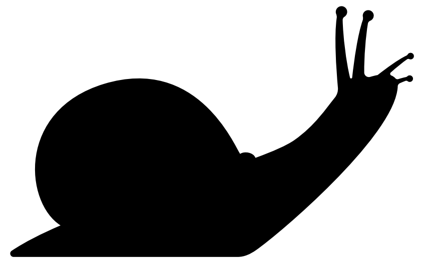
Multiplying by a random matrix of size $\Omega(n^2)$

Generating and summing unit vectors

Making the PRG more Efficient

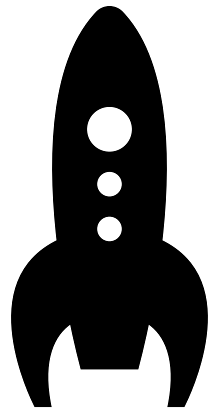
$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto$





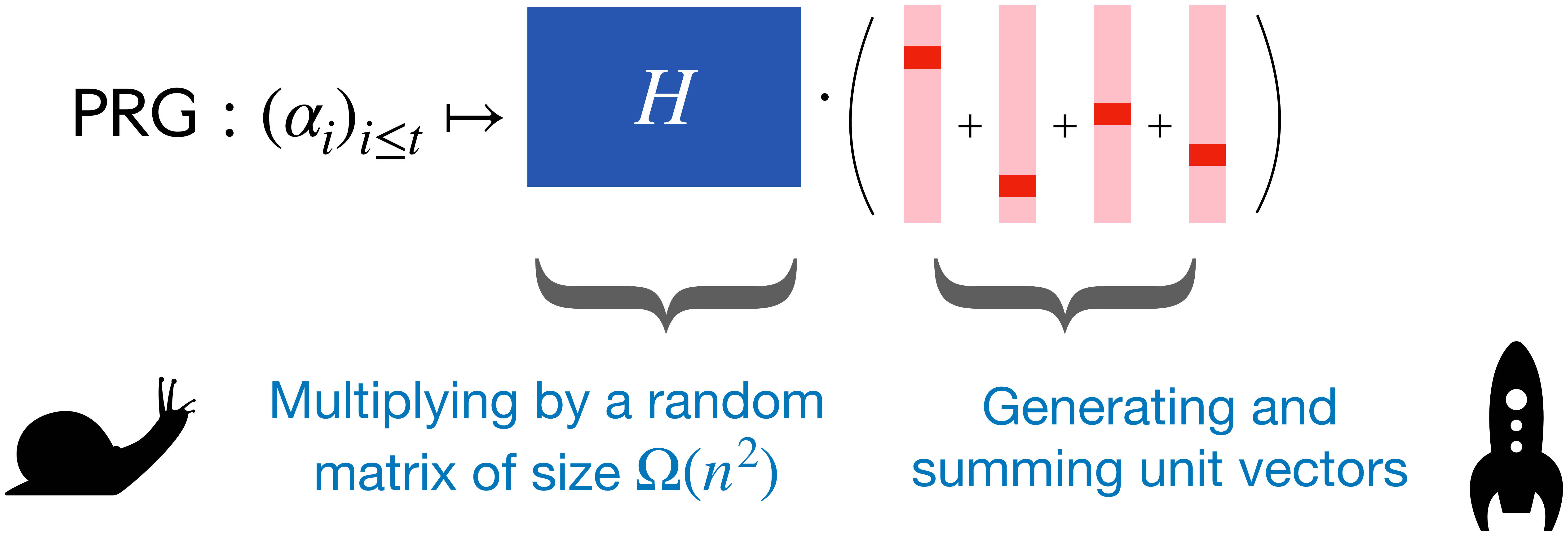
Multiplying by a random matrix of size $\Omega(n^2)$

Generating and summing unit vectors



⚠ n is the total amount of correlated randomness we want to generate! (Think: $n \sim 2^{30}$)

Making the PRG more Efficient



PRG : $(\alpha_i)_{i\leq t} \mapsto H \cdot \left(+ + + \right)$

Multiplying by a random matrix of size $\Omega(n^2)$

Generating and summing unit vectors

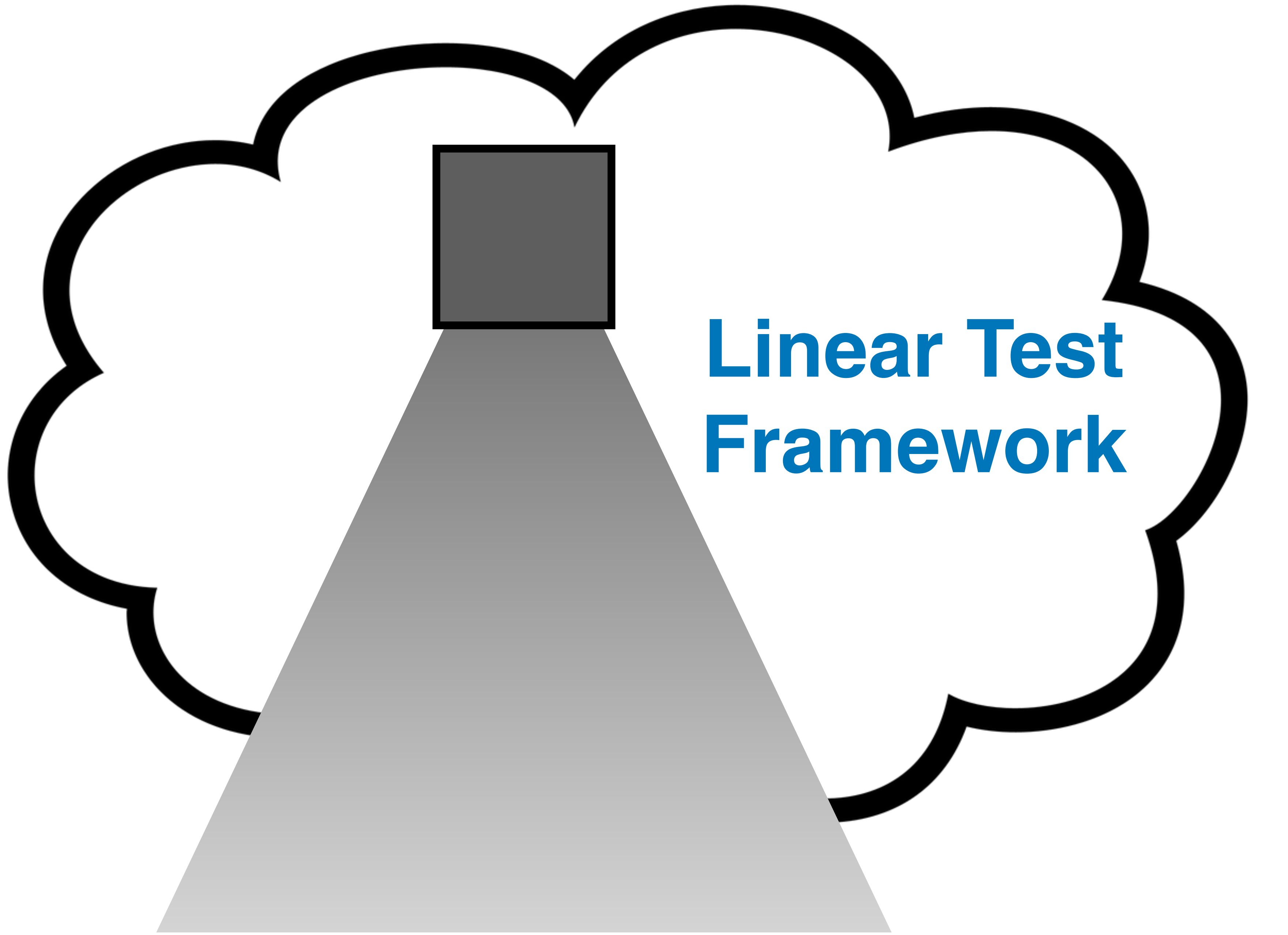
⚠ n is the total amount of correlated randomness we want to generate! (Think: $n \sim 2^{30}$)

Can we replace H with a matrix that allows for fast matrix-vector product?

We need a rule of thumb to know which matrices will yield *plausible* variants of LPN

Security of (variants of) LPN - Linear Tests

A tremendous number of attacks on LPN have been published…

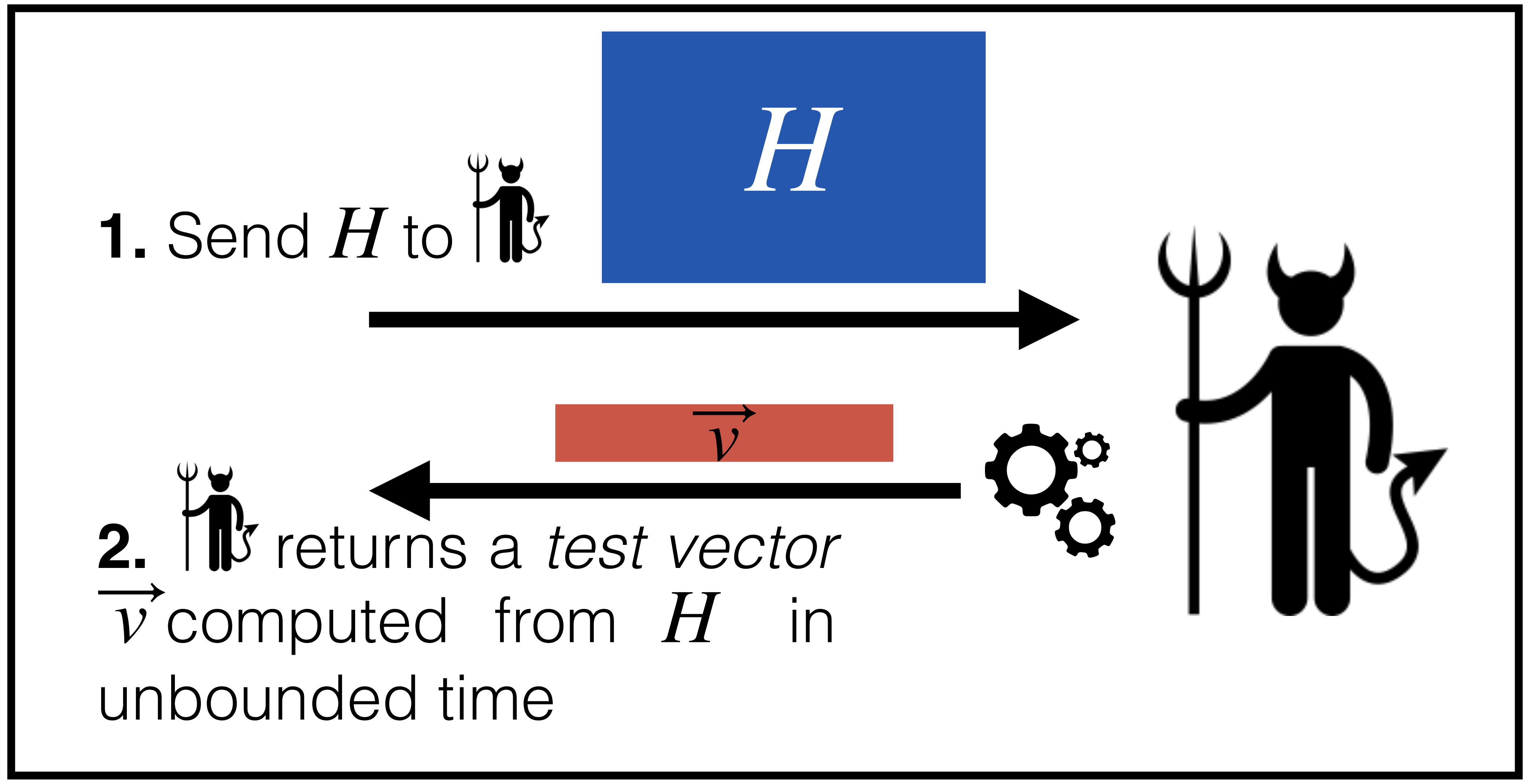


Linear Test Framework

- **Gaussian Elimination attacks**
  - Standard gaussian elimination
  - Blum-Kalai-Wasserman [J.ACM:BKW03]
  - Sample-efficient BKW [A-R:Lyu05]
  - Pooled Gauss [CRYPTO:EKM17]
  - Well-pooled Gauss [CRYPTO:EKM17]
  - Leviel-Fouque [SCN:LF06]
  - Covering codes [JC:GJL19]
  - Covering codes+ [BTV15]
  - Covering codes++ [BV:AC16]
  - Covering codes+++ [EC:ZJW16]
- **Statistical Decoding Attacks**
  - Jabri's attack [ICCC:Jab01]
  - Overbeck's variant [ACISP:Ove06]
  - FKI's variant [Trans.IT:FKI06]
  - Debris-Tillich variant [ISIT:DT17]
- **Information Set Decoding Attacks**
  - Prange's algorithm [Prange62]
  - Stern's variant [ICIT:Stern88]
  - Finiasz and Sendrier's variant [AC:FS09]
  - BJMM variant [EC:BJMM12]
  - May-Ozerov variant [EC:MO15]
  - Both-May variant [PQC:BM18]
  - MMT variant [AC:MMT11]
  - Well-pooled MMT [CRYPTO:EKM17]
  - BLP variant [CRYPTO:BLP11]
- **Other Attacks**
  - Generalized birthday [CRYPTO:Wag02]
  - Improved GBA [Kirchner11]
  - Linearization [EC:BM97]
  - Linearization 2 [INDO:Saa07]
  - Low-weight parity-check [Zichron17]
  - Low-deg approx [ITCS:ABGKR17]

Crucial observation: most attacks fit in the same framework, the *linear test framework. (*)*

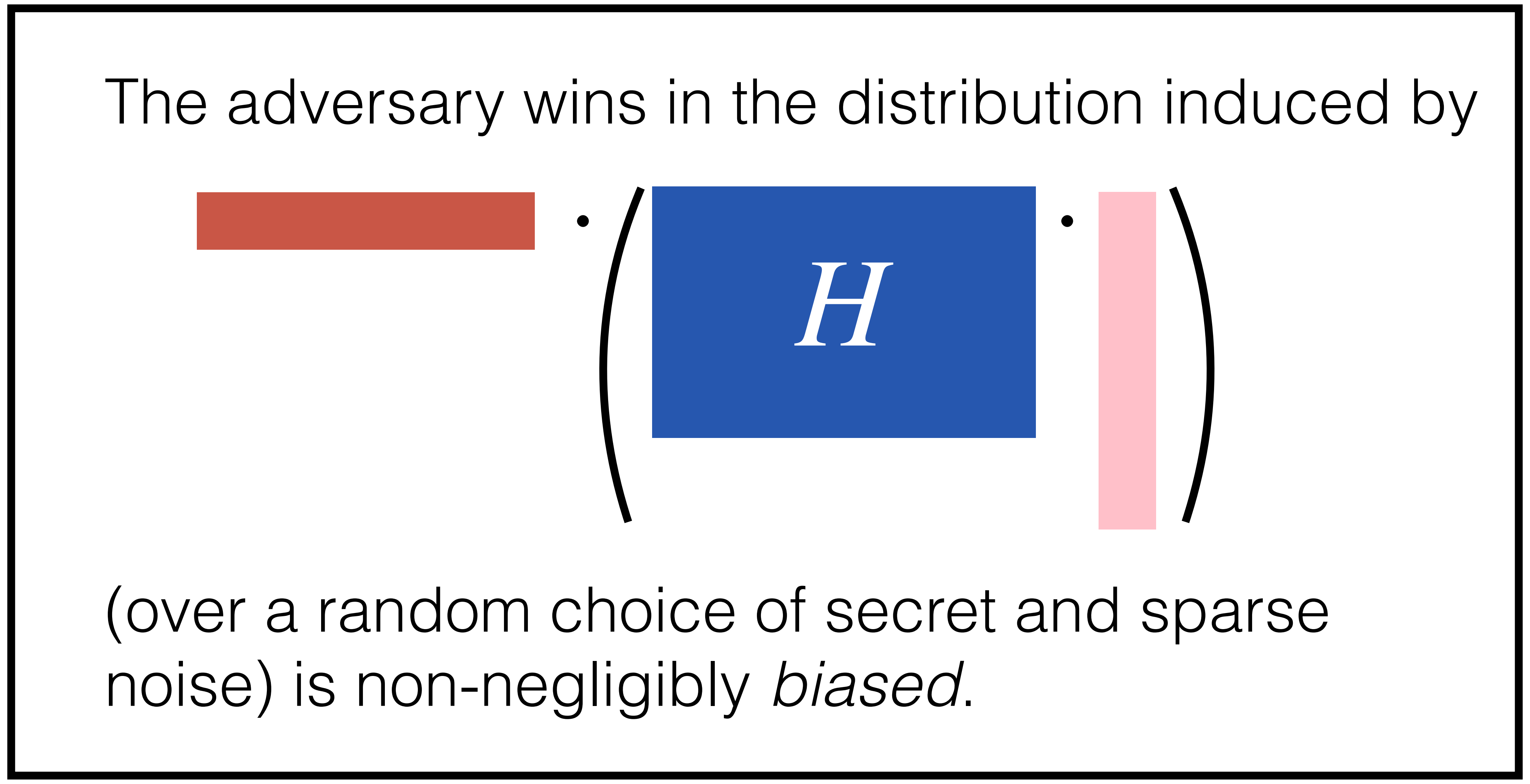
Game



1. Send H to 😈

2. 😈 returns a *test vector* $\vec{v}$ computed from H in unbounded time

Check

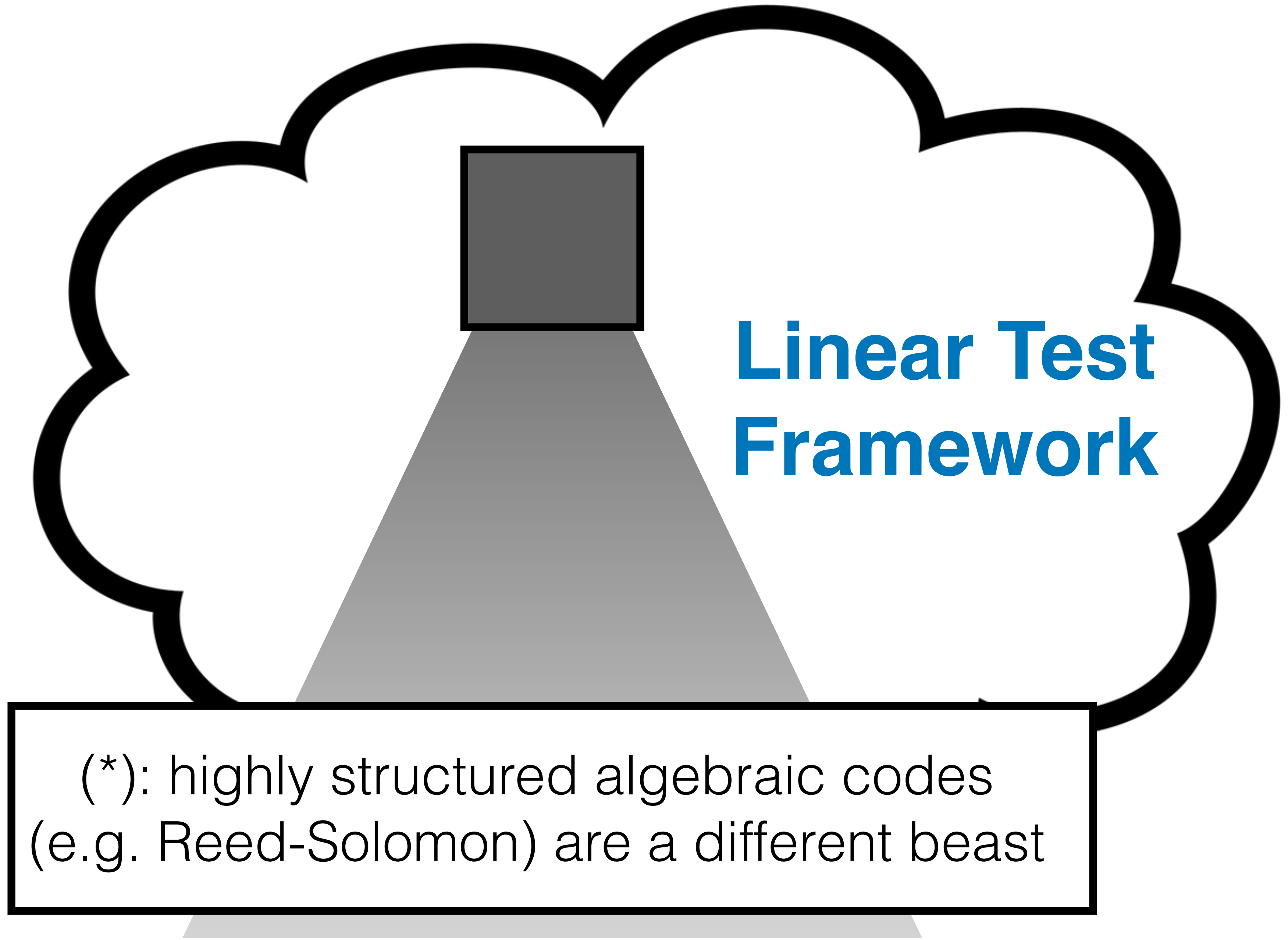


The adversary wins in the distribution induced by

(over a random choice of secret and sparse noise) is non-negligibly *biased*.

Security of (variants of) LPN - Linear Tests

A tremendous number of attacks on LPN have been published…

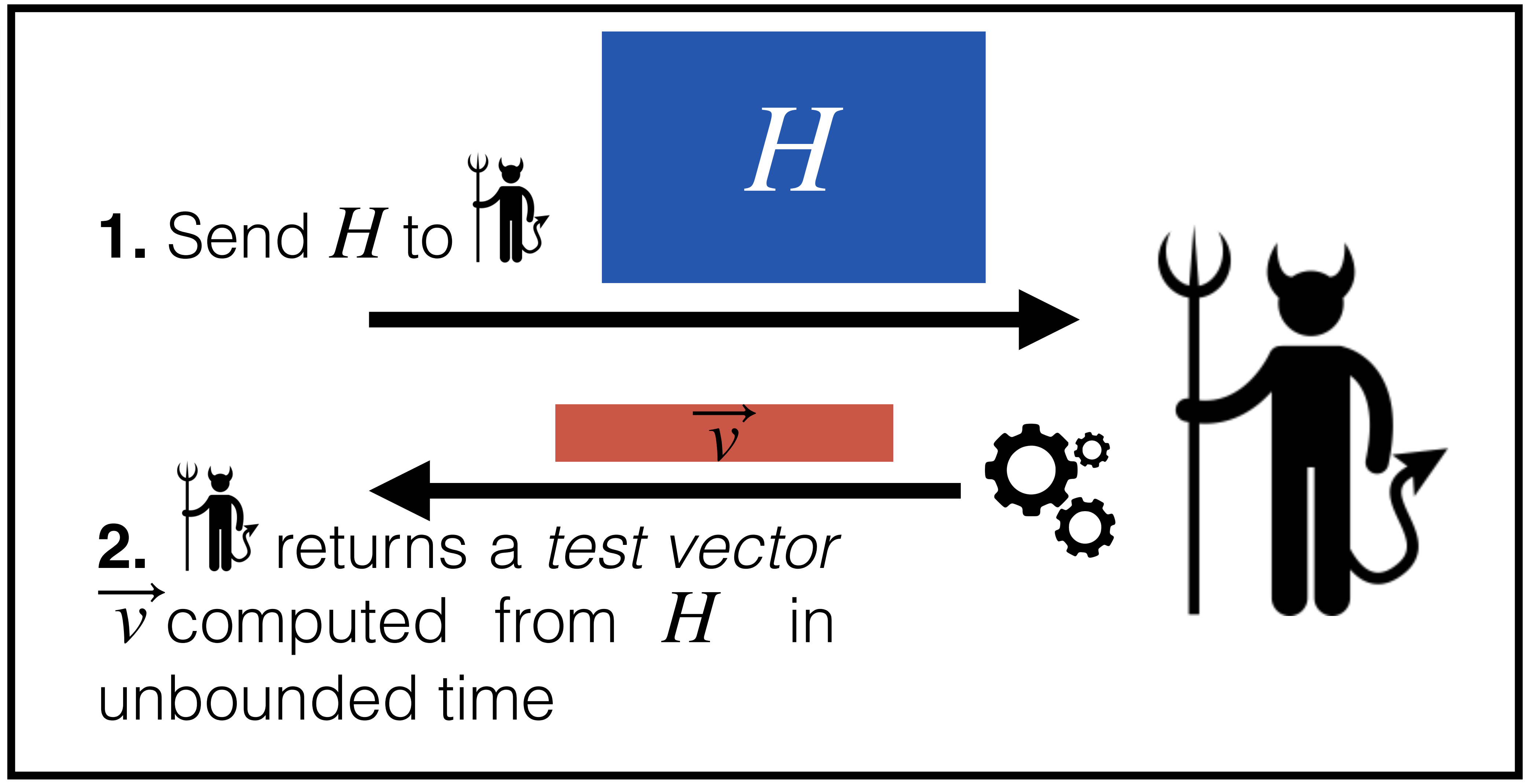


(*): highly structured algebraic codes (e.g. Reed-Solomon) are a different beast

- **Gaussian Elimination attacks**
  - Standard gaussian elimination
  - Blum-Kalai-Wasserman [J.ACM:BKW03]
  - Sample-efficient BKW [A-R:Lyu05]
  - Pooled Gauss [CRYPTO:EKM17]
  - Well-pooled Gauss [CRYPTO:EKM17]
  - Leviel-Fouque [SCN:LF06]
  - Covering codes [JC:GJL19]
  - Covering codes+ [BTV15]
  - Covering codes++ [BV:AC16]
  - Covering codes+++ [EC:ZJW16]
- **Statistical Decoding Attacks**
  - Jabri's attack [ICCC:Jab01]
  - Overbeck's variant [ACISP:Ove06]
  - FKI's variant [Trans.IT:FKI06]
  - Debris-Tillich variant [ISIT:DT17]
- **Information Set Decoding Attacks**
  - Prange's algorithm [Prange62]
  - Stern's variant [ICIT:Stern88]
  - Finiasz and Sendrier's variant [AC:FS09]
  - BJMM variant [EC:BJMM12]
  - May-Ozerov variant [EC:MO15]
  - Both-May variant [PQC:BM18]
  - MMT variant [AC:MMT11]
  - Well-pooled MMT [CRYPTO:EKM17]
  - BLP variant [CRYPTO:BLP11]
- **Other Attacks**
  - Generalized birthday [CRYPTO:Wag02]
  - Improved GBA [Kirchner11]
  - Linearization [EC:BM97]
  - Linearization 2 [INDO:Saa07]
  - Low-weight parity-check [Zichron17]
  - Low-deg approx [ITCS:ABGKR17]

Crucial observation: most attacks fit in the same framework, the *linear test framework. (*)*

Game

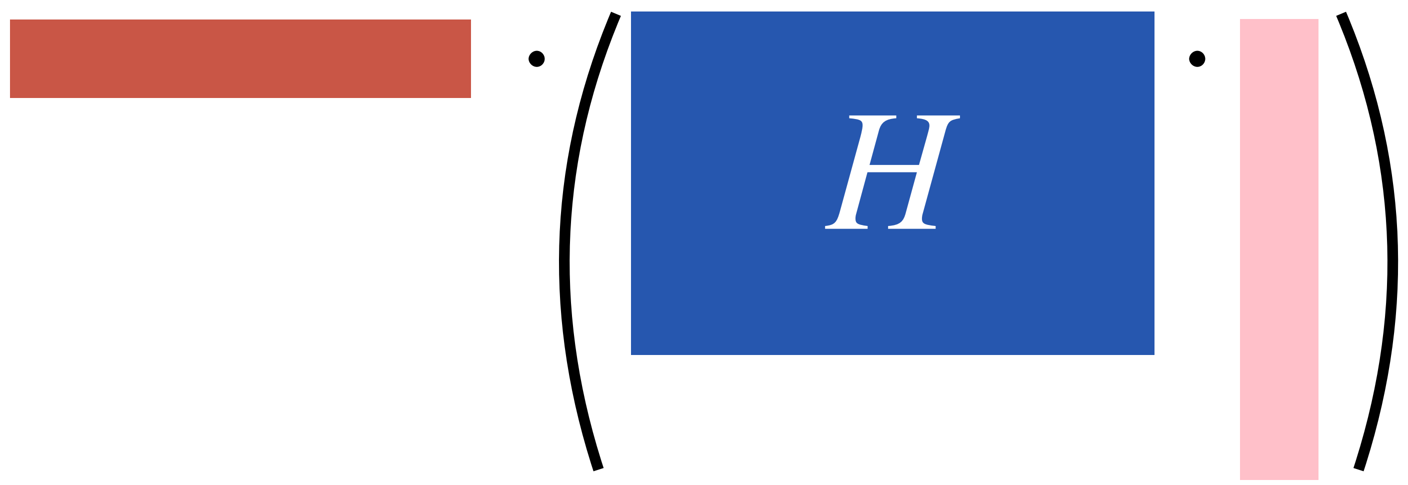


1. Send H to

2. returns a *test vector* $\vec{v}$ computed from H in unbounded time

Check

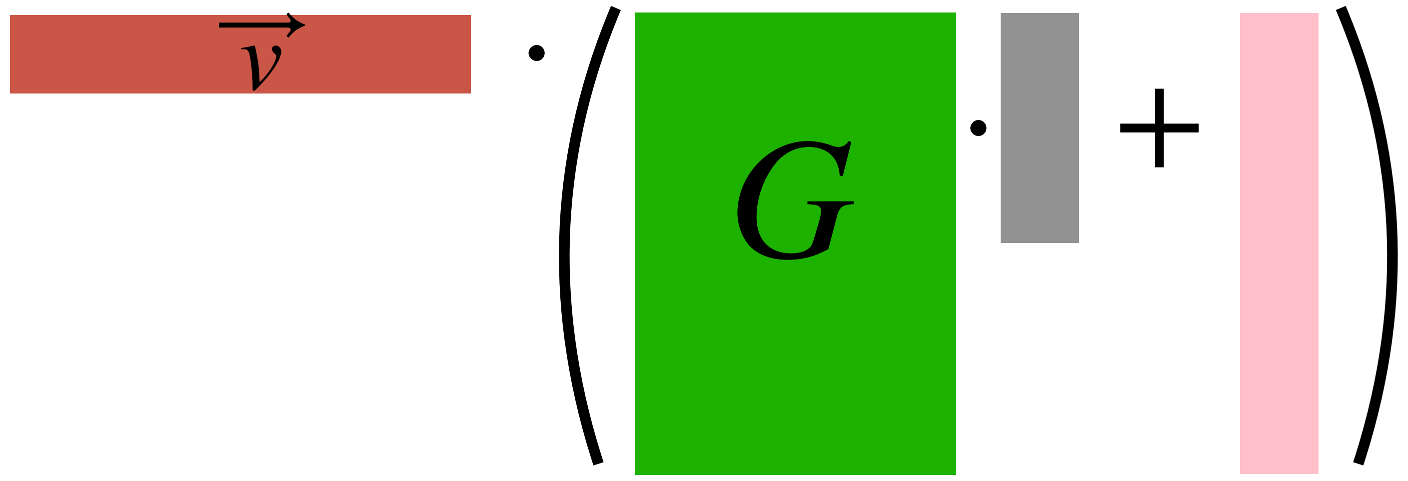
The adversary wins in the distribution induced by



(over a random choice of secret and sparse noise) is non-negligibly *biased*.

Withstanding Linear Tests

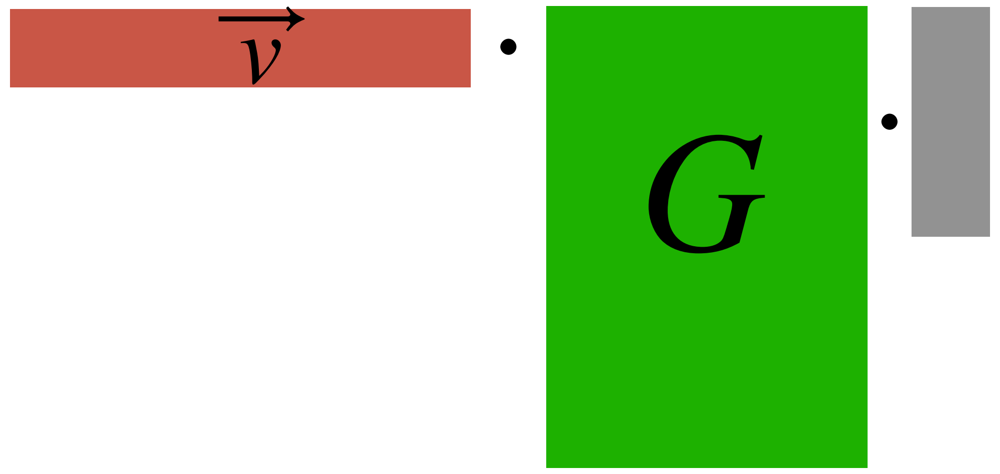
The adversary wins in the distribution induced by



(over a random choice of secret and sparse noise) is non-negligibly *biased*.

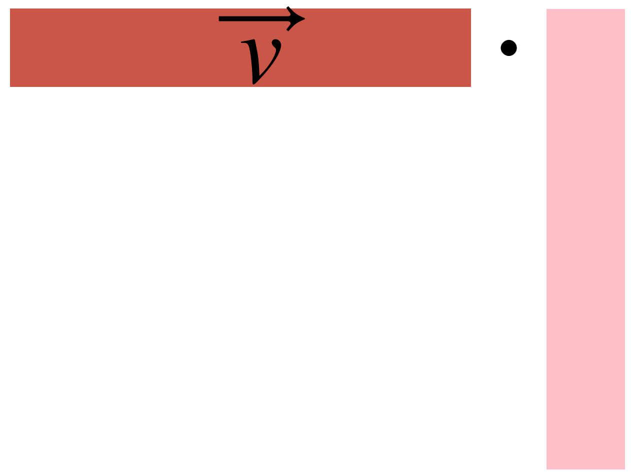
We have a sum of two distributions:

Induced by the *codeword*



Protects against *light* linear tests

Induced by the *noise vector*

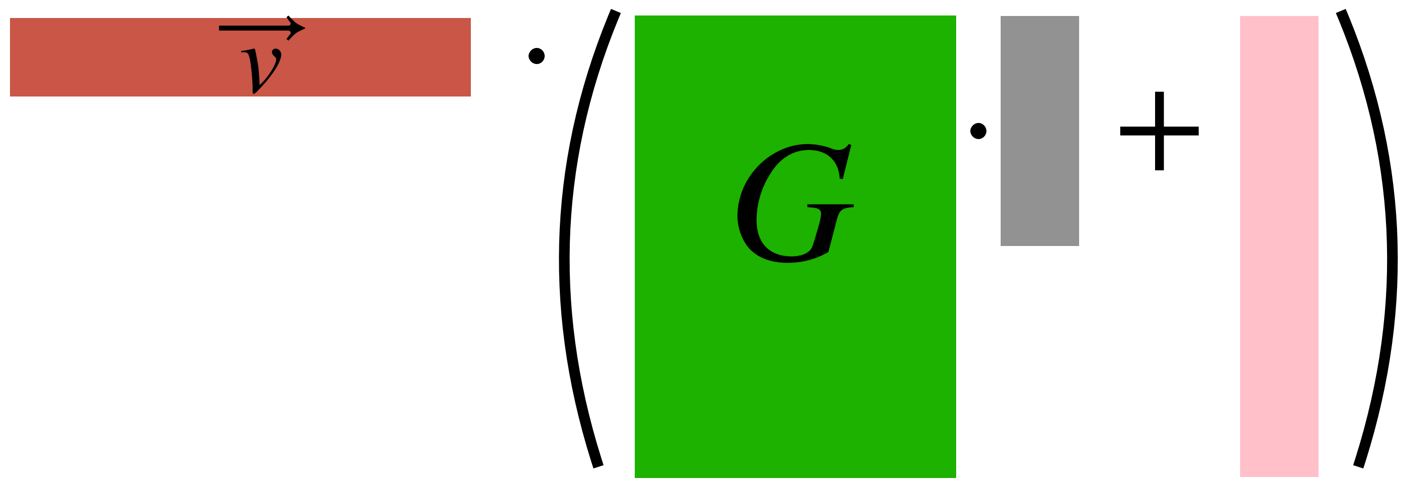


Protects against *heavy* linear tests

Claim: Assume t (number of noisy coordinates) is set to a security parameter. If there is a constant c such that every subset of $c \cdot n$ rows of G is linearly independent, no linear test can distinguish $G \cdot \vec{s} + \vec{e}$ from random.

Withstanding Linear Tests

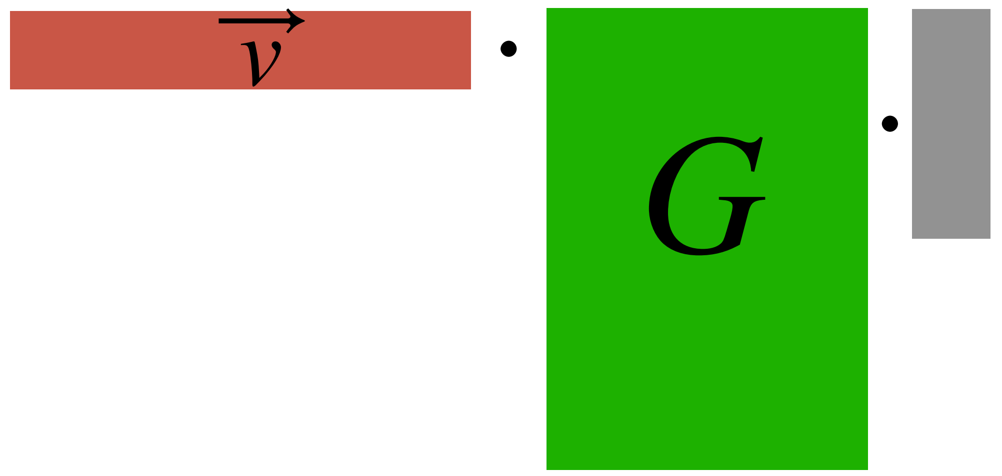
The adversary wins in the distribution induced by



(over a random choice of secret and sparse noise) is non-negligibly *biased*.

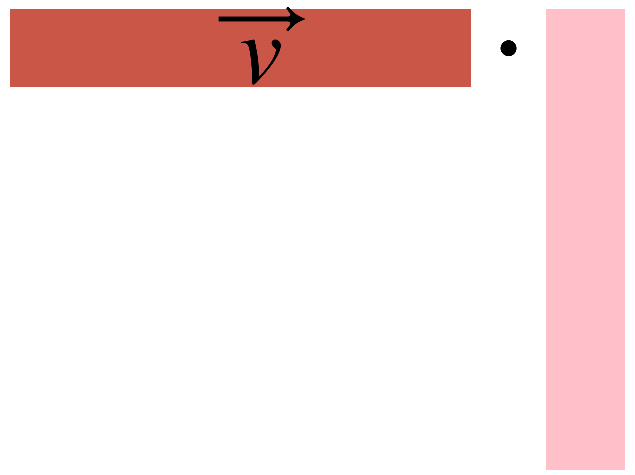
We have a sum of two distributions:

Induced by the *codeword*



Protects against *light* linear tests

Induced by the *noise vector*



Protects against *heavy* linear tests

Claim: Assume t (number of noisy coordinates) is set to a security parameter. If there is a constant c such that every subset of $c \cdot n$ rows of G is linearly independent, no linear test can distinguish $G \cdot \vec{s} + \vec{e}$ from random.

Rephrasing the sufficient condition:

Every subset of $O(n)$ rows of G is linearly independent
$\Longleftrightarrow$ the left-kernel of G does not contain nonzero vector of weight less than $O(n)$
$\Longleftrightarrow$ the *dual code* of G, i.e., the code generated by *the transpose of its parity check matrix* H, has linear minimum distance

Pseudorandom Correlation Generators - Efficiency

Goal: computing  fast, such that the code  is LPN-friendy

We want to find a matrix $M = H^\intercal$ such that (1) the code generated by M is a good code, and (2) computing ~~$M^\intercal \cdot \vec{v}$~~ takes time $O(n)$ for any $\vec{v}$

$M \cdot \vec{v}$ (this is the *transposition principle*)

$\Longrightarrow$ We need to find a *good* and *linear-time encodable* code. And we want it concretely efficient!

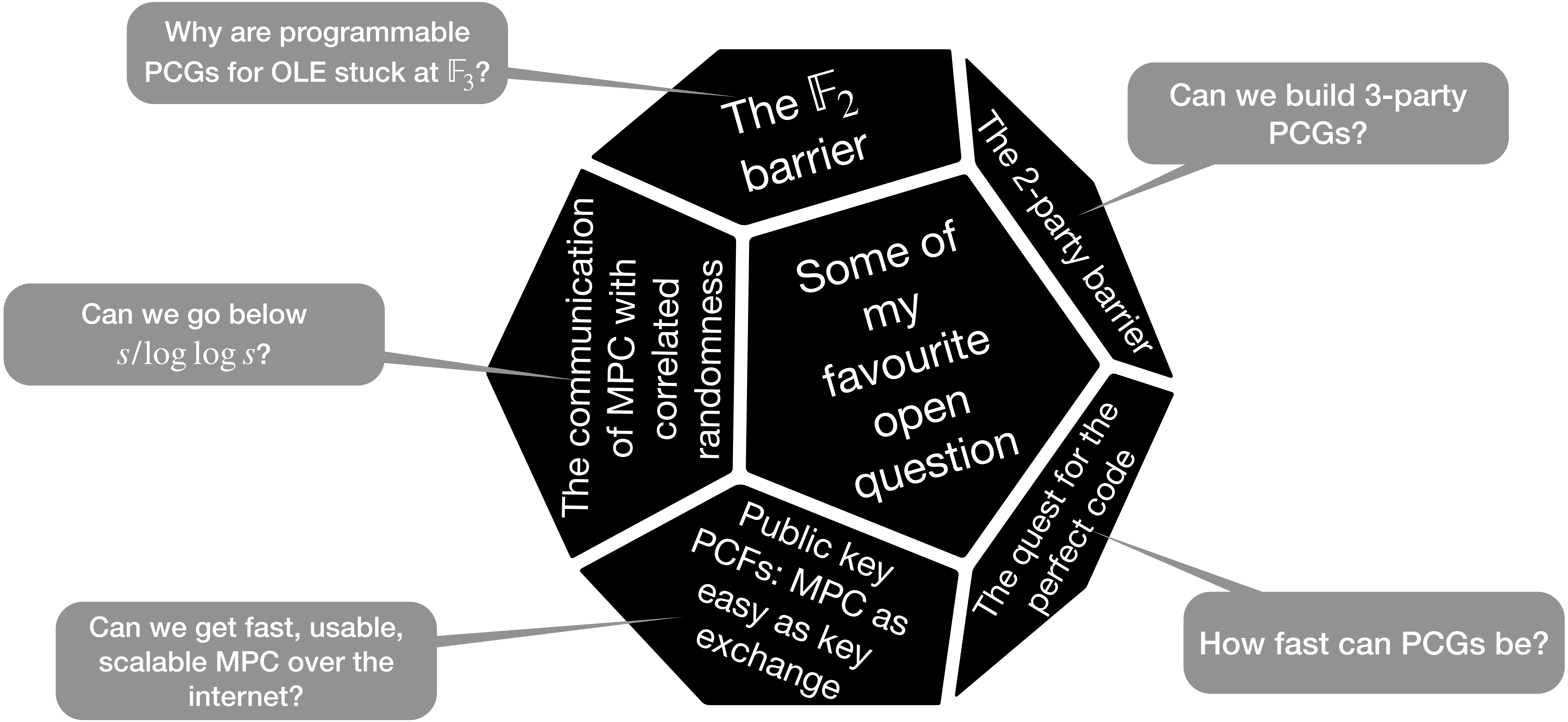
Pseudorandom Correlation Generators - Efficiency

There is an ongoing and exciting quest for pinpointing the *right* code for PCG applications:

- CCS:Boyle-C-Gilboa-Ishai'18 suggested using LDPC code
- CCS:Boyle-C-Gilboa-Ishai-Kohl-Rindal-Scholl'19 moved to quasi-cyclic codes

  due to concern regarding linear-time encoding of LDPC codes
- Crypto:C-Raghuraman-Rindal'21: tailored LDPC with heuristic & experimental support
- Crypto:Boyle-C-Gilboa-Ishai-Kohl-Resch—Scholl'22: Expand-Accumulate codes
- Latest news: there's apparently a new proposal that suggests Expand-Convolute codes instead (and which breaks Silver along the way!)
- There are a few more codes I'd like to investigate, the quest continues!

Some of my Favourite Open Questions



Some of my favourite open question

The $\mathbb{F}_2$ barrier

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The 2-party barrier

Can we build 3-party PCGs?

The quest for the perfect code

How fast can PCGs be?

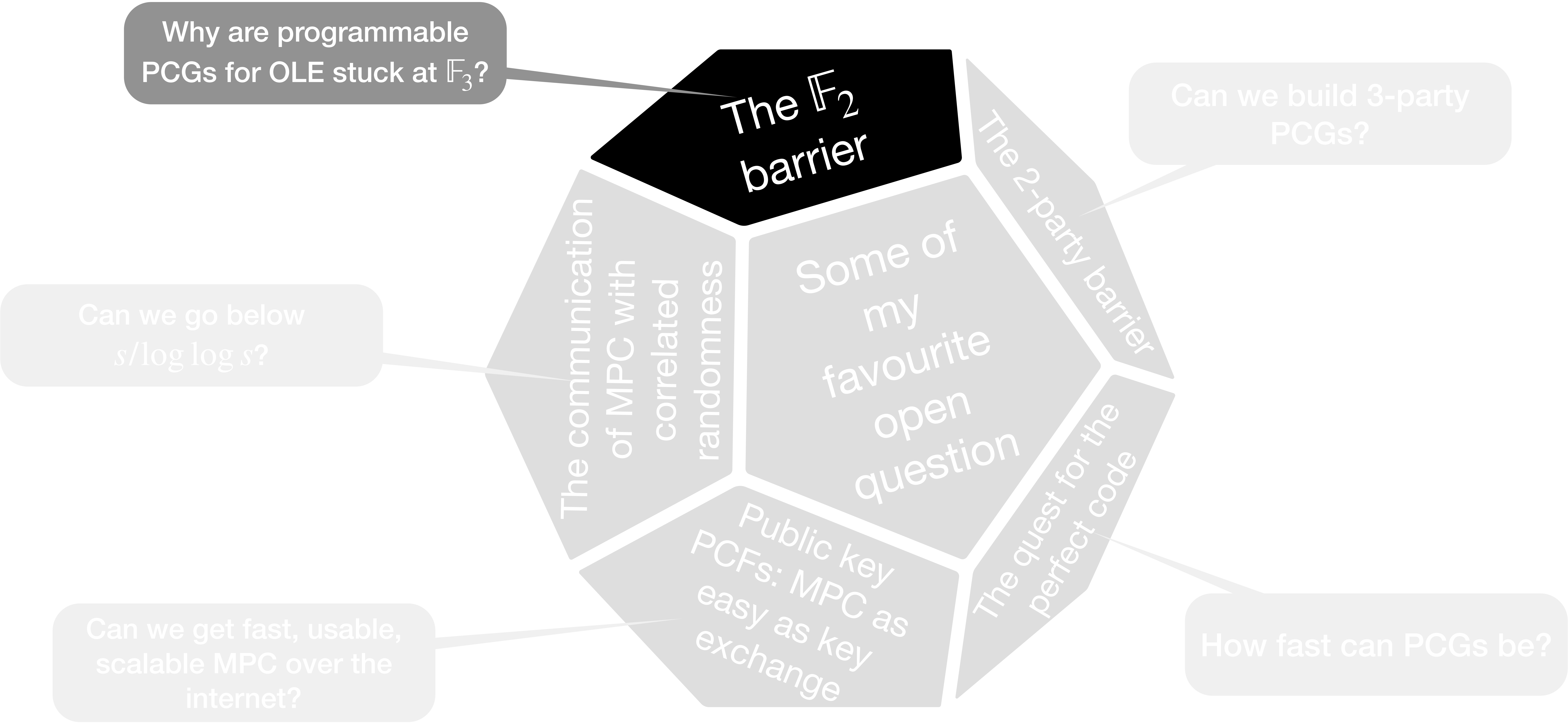
Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

Some of my Favourite Open Questions



Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The $\mathbb{F}_2$ barrier

The 2-party barrier

Can we build 3-party PCGs?

Some of my favourite open question

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The quest for the perfect code

How fast can PCGs be?

OLE Correlations

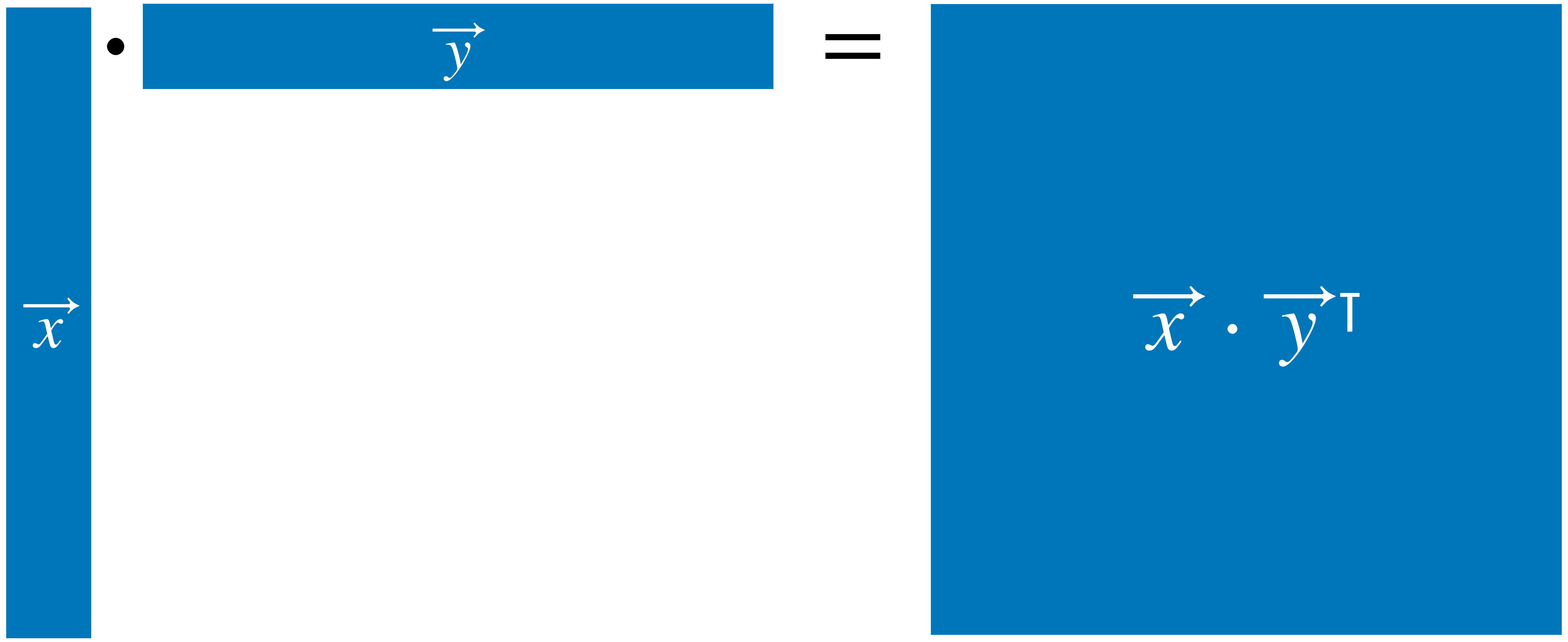
OLE over $\mathbb{F}$ is the type of correlation we want to do (semi-honest) secure computation of arithmetic circuits over $\mathbb{F}$.

In an OLE, Alice gets $a \leftarrow \mathbb{F}$, Bob gets $b \leftarrow \mathbb{F}$, and Alice and Bob get random shares of $a \cdot b$.

OLE Correlations, the LPN Way

Goal:

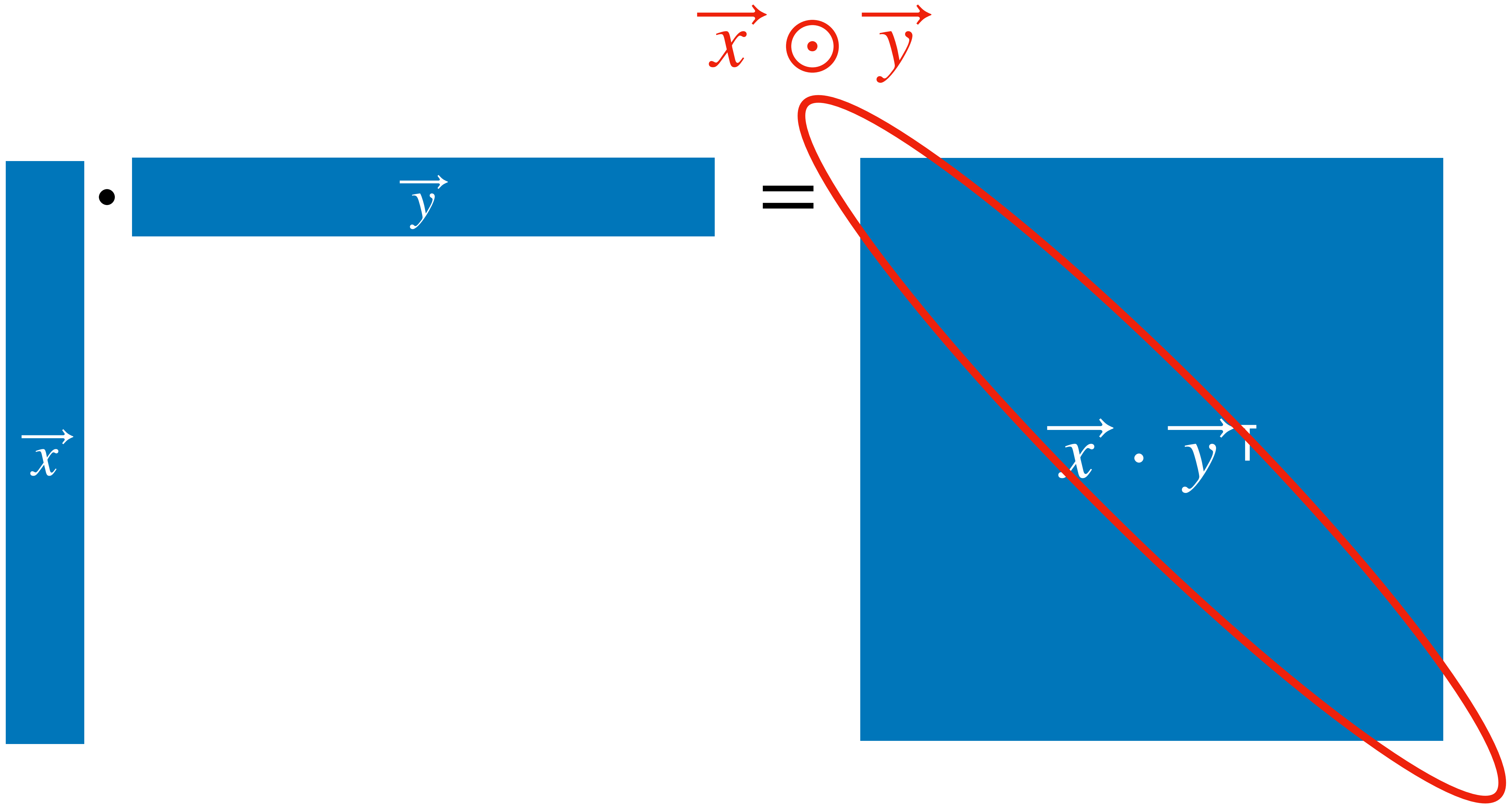
- Alice gets a pseudorandom vector $\overrightarrow{x}$
- Bob gets a pseudorandom vector $\overrightarrow{y}$
- Alice and Bob get shares of $\overrightarrow{x} \odot \overrightarrow{y}$



OLE Correlations, the LPN Way

Goal:

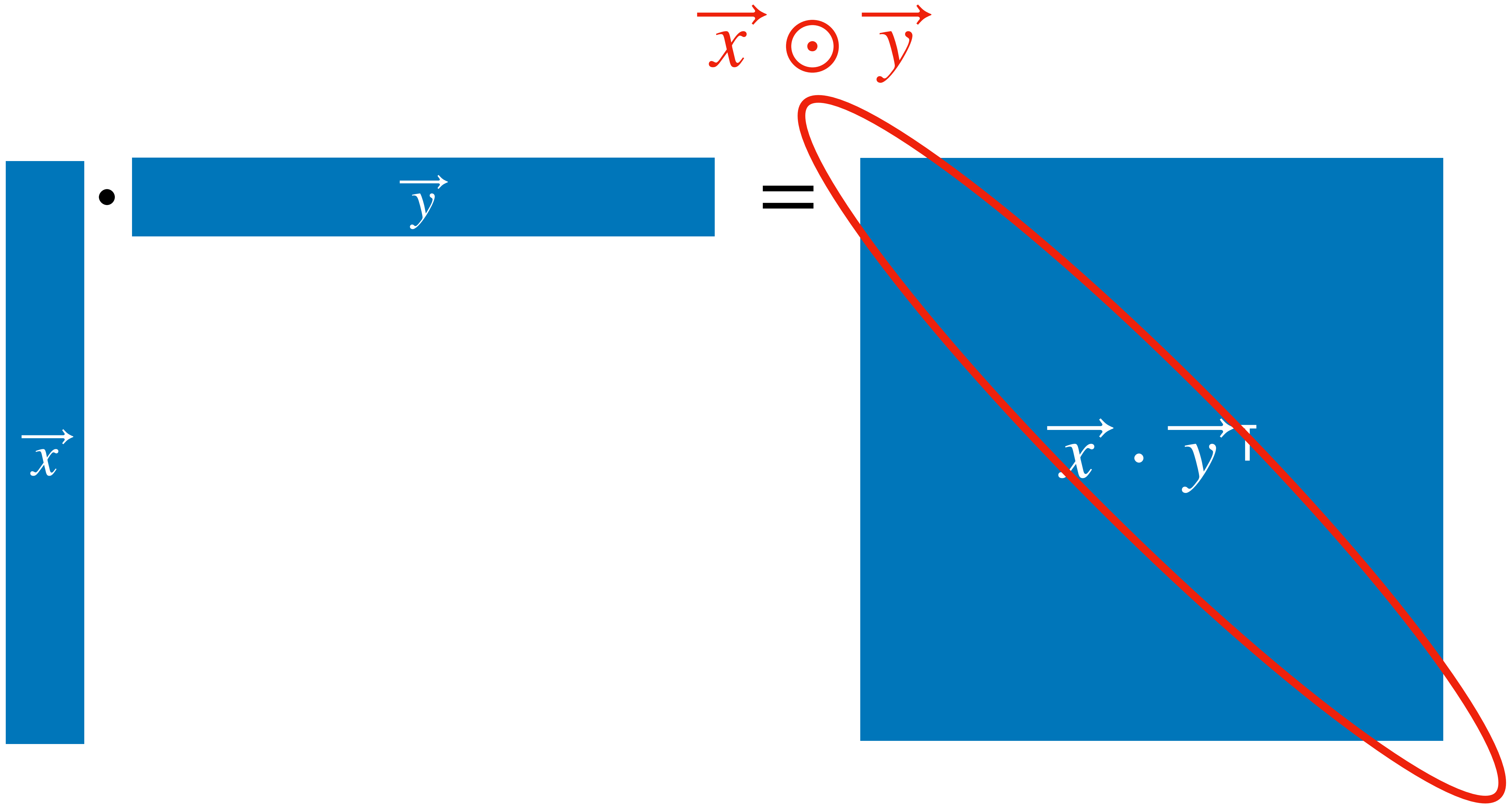
- Alice gets a pseudorandom vector $\overrightarrow{x}$
- Bob gets a pseudorandom vector $\overrightarrow{y}$
- Alice and Bob get shares of $\overrightarrow{x} \odot \overrightarrow{y}$



OLE Correlations, the LPN Way

Goal:

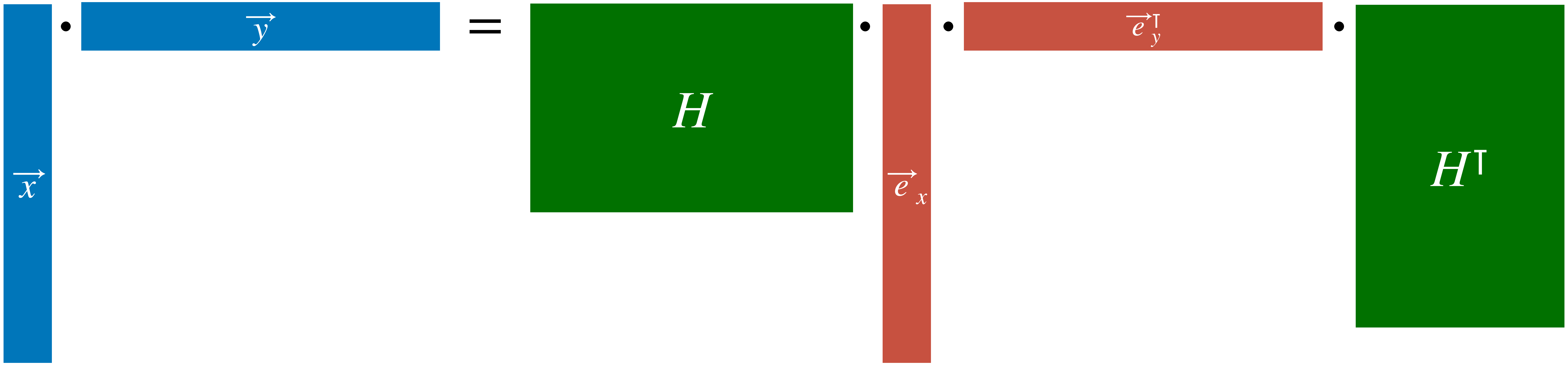
- Alice gets a pseudorandom vector $\overrightarrow{x} = H \cdot \overrightarrow{e}_x$
- Bob gets a pseudorandom vector $\overrightarrow{y} = H \cdot \overrightarrow{e}_y$
- Alice and Bob get shares of $\overrightarrow{x} \odot \overrightarrow{y}$



OLE Correlations, the LPN Way

Goal:

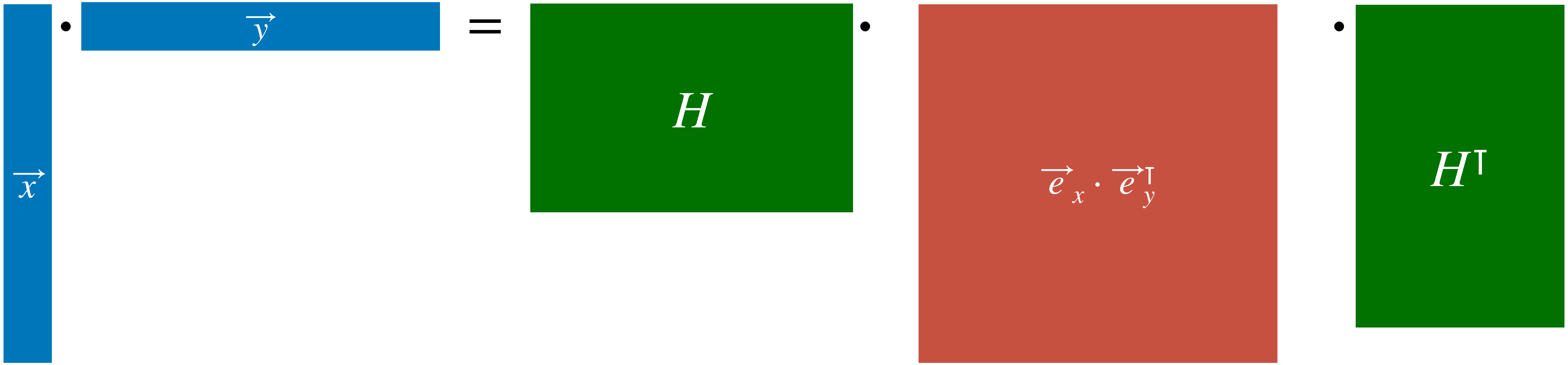
- Alice gets a pseudorandom vector $\overrightarrow{x} = H \cdot \overrightarrow{e}_x$
- Bob gets a pseudorandom vector $\overrightarrow{y} = H \cdot \overrightarrow{e}_y$
- Alice and Bob get shares of $\overrightarrow{x} \odot \overrightarrow{y}$



OLE Correlations, the LPN Way

Goal:

- Alice gets a pseudorandom vector $\overrightarrow{x} = H \cdot \overrightarrow{e}_x$
- Bob gets a pseudorandom vector $\overrightarrow{y} = H \cdot \overrightarrow{e}_y$
- Alice and Bob get shares of $\overrightarrow{x} \odot \overrightarrow{y}$



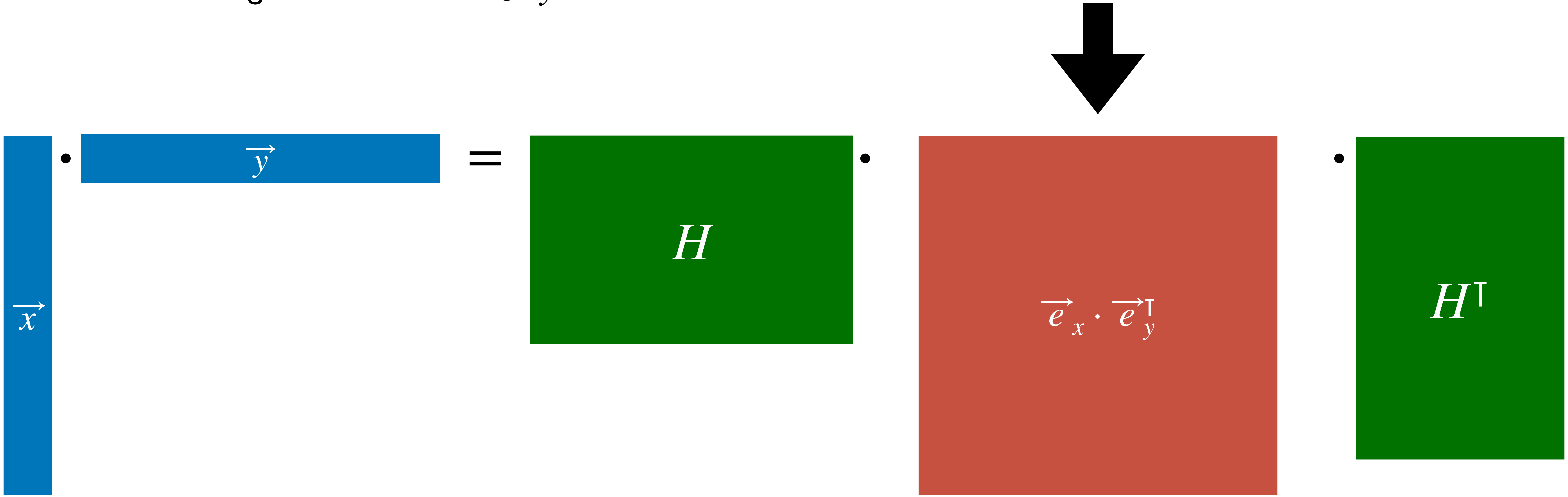
OLE Correlations, the LPN Way

Goal:

- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$

This is a t^2-sparse matrix, *i.e.* a sum of t^2 point functions!

$\Longrightarrow$ can be generated with comm. $O(\lambda t^2 \log n)$



$$\vec{x} \cdot \vec{y} = H \cdot \vec{e}_x \cdot \vec{e}_y^\intercal \cdot H^\intercal$$

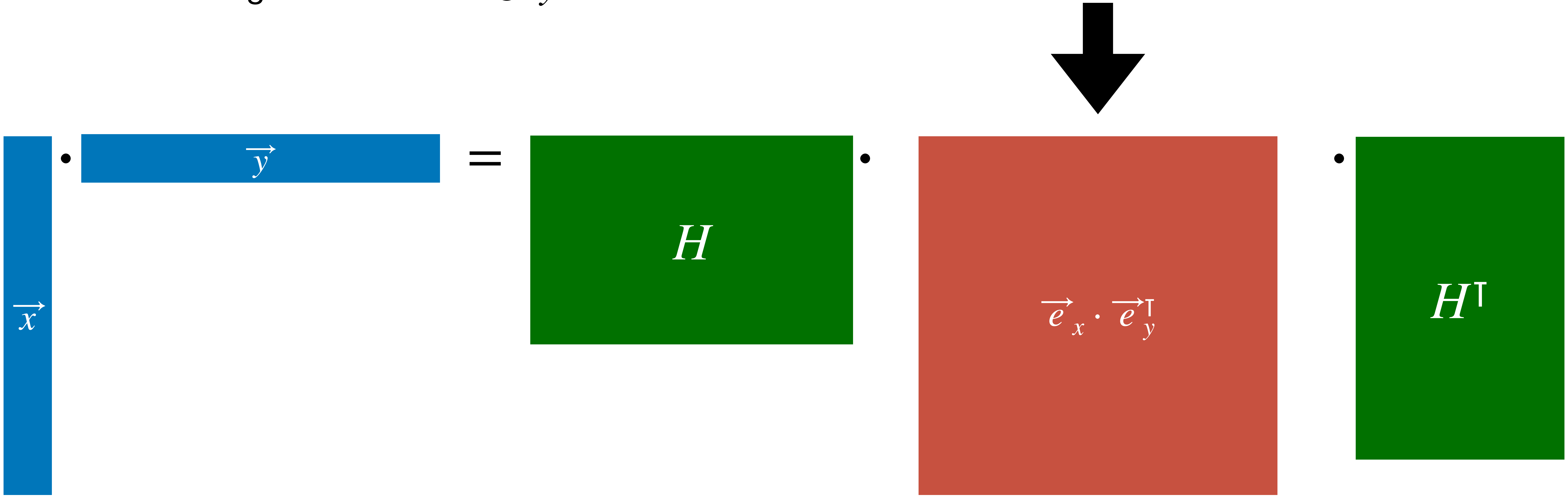
OLE Correlations, the LPN Way

Goal:

- Alice gets a pseudorandom vector $\vec{x} = H \cdot \vec{e}_x$
- Bob gets a pseudorandom vector $\vec{y} = H \cdot \vec{e}_y$
- Alice and Bob get shares of $\vec{x} \odot \vec{y}$

This is a t^2-sparse matrix, *i.e.* a sum of t^2 point functions!

$\Longrightarrow$ can be generated with comm. $O(\lambda t^2 \log n)$



Uses only LPN

Costs $\omega(n^2)$! (Think: $n \sim 2^{30}\ldots$)

OLE Correlations, the *Ring*-LPN Way

Crypto: Boyle-C-Gilboa-Ishai-Kohl-Scholl'20

Let $\mathscr{R}$ be the ring $\mathbb{Z}_p/F(X)$ where $F(X)$ is a degree-n polynomial that splits entirely, and $p > n$.

Ring-LPN assumption: $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathscr{R}$ and (e, f) are random t-*sparse* polynomials.

Observation: we can get n OLE correlations from a single 'ring-OLE' correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathscr{R}$: the OLE correlations are obtained by reducing x, y, and $x \cdot y$ modulo each of the linear factors F_i of F.

Construction:

- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where (e_x, f_x) are t-sparse polynomials over $\mathscr{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t-sparse polynomials over $\mathscr{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials a^2, a are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all t^2-sparse polynomials

OLE Correlations, the *Ring*-LPN Way

Crypto: Boyle-C-Gilboa-Ishai-Kohl-Scholl'20

Let $\mathscr{R}$ be the ring $\mathbb{Z}_p/F(X)$ where $F(X)$ is a degree-n polynomial that splits entirely, and $p > n$.

Ring-LPN assumption: $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathscr{R}$ and (e, f) are random t-*sparse* polynomials.

Observation: we can get n OLE correlations from a single 'ring-OLE' correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathscr{R}$: the OLE correlations are obtained by reducing $x, y,$ and $x \cdot y$ modulo each of the linear factors F_i of F.

Construction:

- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where (e_x, f_x) are t-sparse polynomials over $\mathscr{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t-sparse polynomials over $\mathscr{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials a^2, a are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all t^2-sparse polynomials

☺ Costs only $O(n \cdot \log n)$

☹

- 'Splittable ring-LPN' deserves further study

OLE Correlations, the *Ring*-LPN Way

Crypto: Boyle-C-Gilboa-Ishai-Kohl-Scholl'20

Let $\mathscr{R}$ be the ring $\mathbb{Z}_p/F(X)$ where $F(X)$ is a degree-n polynomial that splits entirely, and $p > n$.

Ring-LPN assumption: $(a, b) \sim (a, a \cdot e + f)$ where $(a, b) \leftarrow \mathscr{R}$ and (e, f) are random t-*sparse* polynomials.

Observation: we can get n OLE correlations from a single 'ring-OLE' correlation $(x, y, \langle x \cdot y \rangle)$ over $\mathscr{R}$: the OLE correlations are obtained by reducing $x, y,$ and $x \cdot y$ modulo each of the linear factors F_i of F.

Construction:

- Alice gets a pseudorandom polynomial $x = a \cdot e_x + f_x$ where (e_x, f_x) are t-sparse polynomials over $\mathscr{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t-sparse polynomials over $\mathscr{R}$
- Alice and Bob get shares of $x \cdot y = a^2 \cdot (e_x e_y) + a \cdot (e_x f_y + f_x e_y) + f_x f_y$

The polynomials a^2, a are public, and $e_x e_y, e_x f_y, f_x e_y, f_x f_y$ are all t^2-sparse polynomials

Costs only $O(n \cdot \log n)$

- 'Splittable ring-LPN' deserves further study
- $\mathbb{F}$ must be large!

OLE Correlations, from Quasi-Abelian Syndrome Decoding

How do we break this ‘field-size barrier’? An answer in our recent Crypto paper (Bombar-C-Couvreur-Ducros’23): we move to *quasi-abelian* codes, which are defined over group algebras.

High level intuition

The group algebra structure gives a suitable framework to find the *right* polynomial P to instantiate an LPN variant over a ring $\mathscr{R} = \mathbb{F}[X_1, \cdots, X_d]/P(X_1, \cdots, X_d)$ such that

- $\mathscr{R} \sim \mathbb{F} \times \cdots \times \mathbb{F}$ (*i.e.* we get many copies of an OLE over $\mathbb{F}$)
- The underlying assumption is plausibly secure (*i.e.* resists linear attacks)

Using multivariate rings gives us many more roots of P even for a small $\mathbb{F}$! In fact, we can get up to $(|\mathbb{F}| - 1)^d$ copies of an OLE over $\mathbb{F}$.

OLE Correlations, from Quasi-Abelian Syndrome Decoding

How do we break this 'field-size barrier'? An answer in our recent Crypto paper (Bombar-C-Couvreur-Ducros'23): we move to *quasi-abelian* codes, which are defined over group algebras.

High level intuition

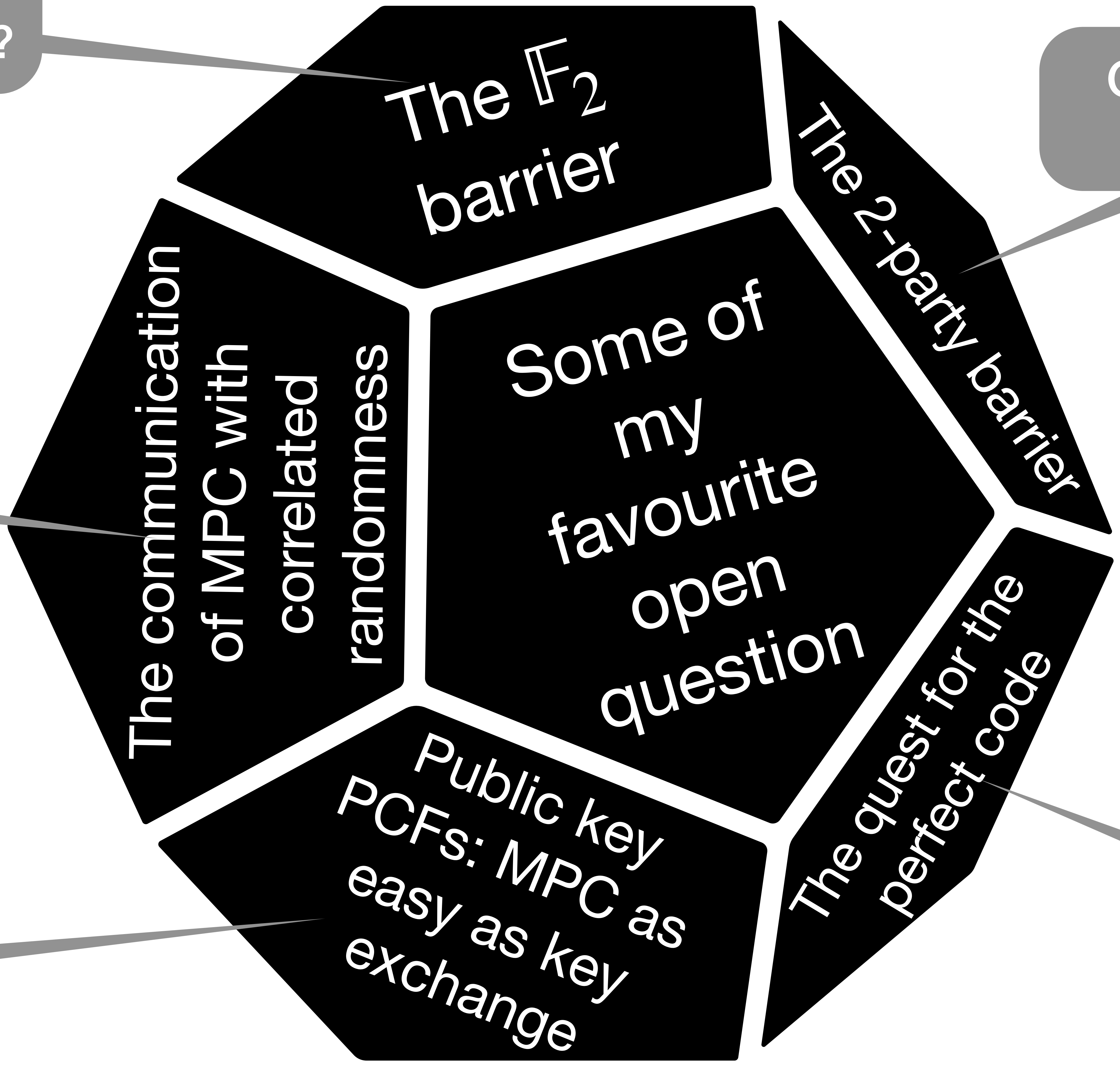
The group algebra structure gives a suitable framework to find the *right* polynomial P to instantiate an LPN variant over a ring $\mathscr{R} = \mathbb{F}[X_1, \cdots, X_d]/P(X_1, \cdots, X_d)$ such that

- $\mathscr{R} \sim \mathbb{F} \times \cdots \times \mathbb{F}$ (*i.e.* we get many copies of an OLE over $\mathbb{F}$)
- The underlying assumption is plausibly secure (*i.e.* resists linear attacks)

Using multivariate rings gives us many more roots of P even for a small $\mathbb{F}$! In fact, we can get up to $(\underbrace{|\mathbb{F}| - 1})^d$ copies of an OLE over $\mathbb{F}$.

This only gives something meaningful up to $\mathbb{F}_3$!

Some of my Favourite Open Questions



Some of my favourite open question

The $\mathbb{F}_2$ barrier

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The 2-party barrier

Can we build 3-party PCGs?

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

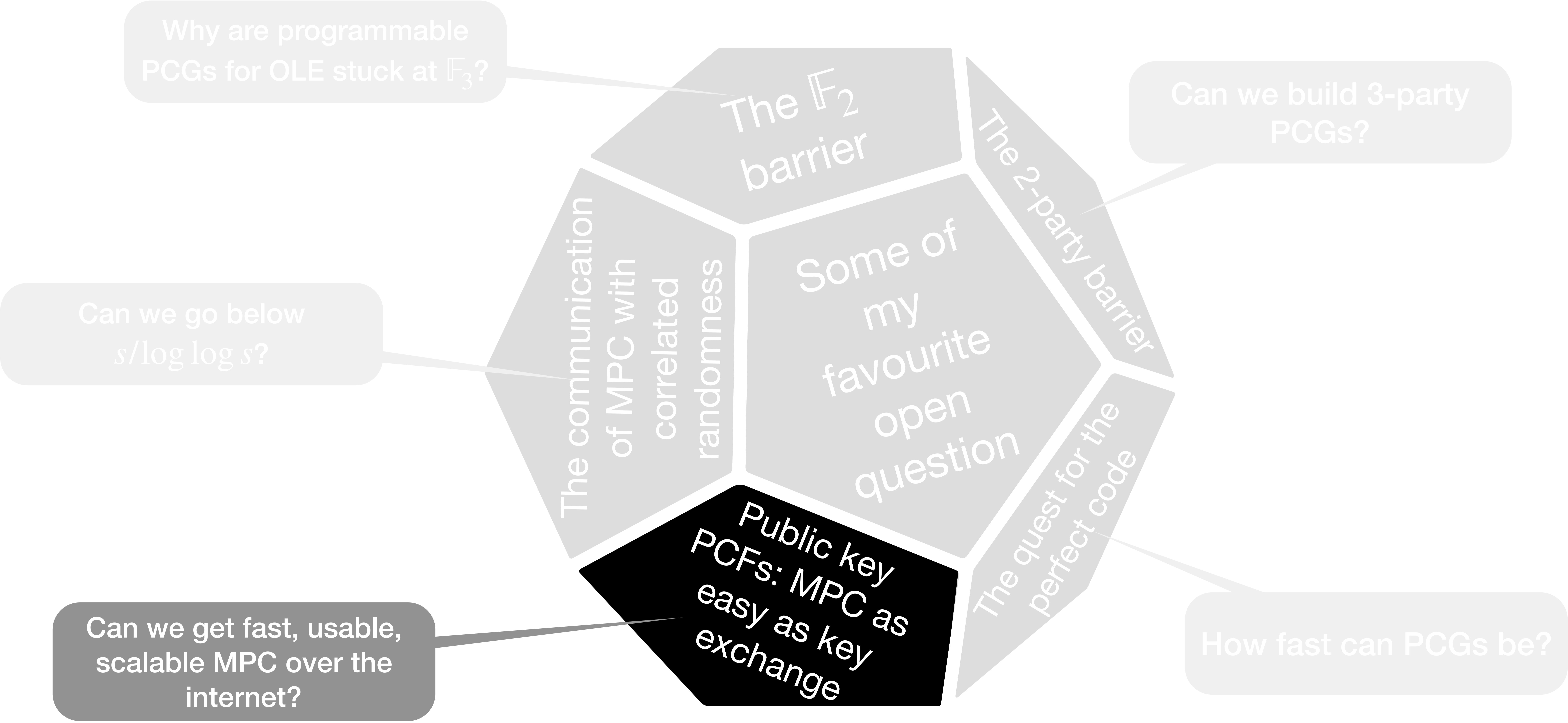
The quest for the perfect code

How fast can PCGs be?

Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

Some of my Favourite Open Questions



Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The $\mathbb{F}_2$ barrier

The 2-party barrier

Can we build 3-party PCGs?

Can we go below $s/\log\log s$?

The communication of MPC with correlated randomness

Some of my favourite open question

The quest for the perfect code

How fast can PCGs be?

Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

A Closer Look at Secure Communication

Our ultimate goal is *practical* MPC that can be deployed and used over the web

Secure communication is already widely deployed and in use

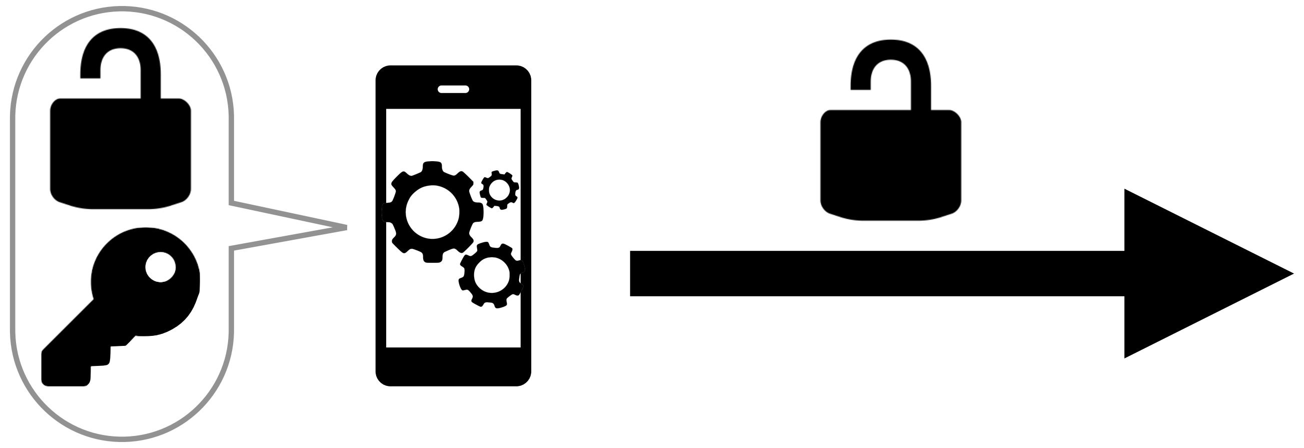
> 85% of the total internet traffic is encrypted

⟹ Let us look at secure communication’s recipe for success!

A Closer Look at Secure Communication

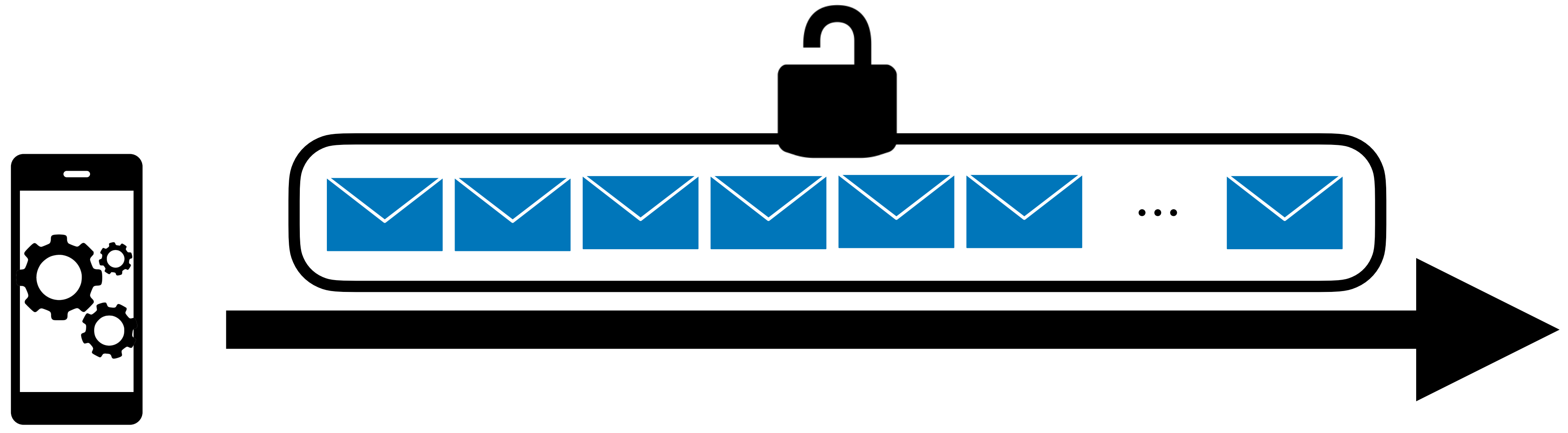
Two Phases:

Key exchange phase



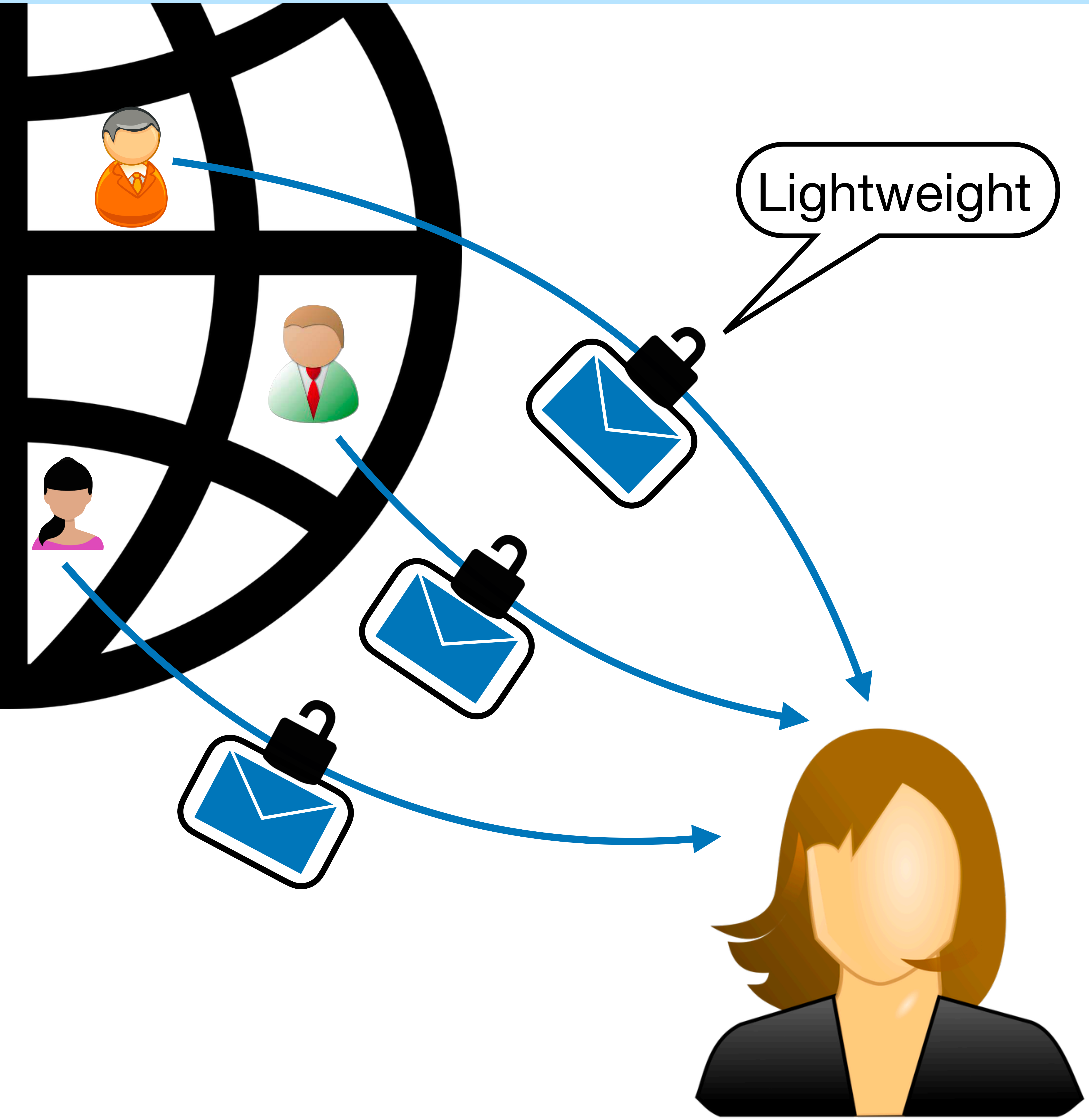
- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot |$🔒$|$ (not n^2)

Encryption phase



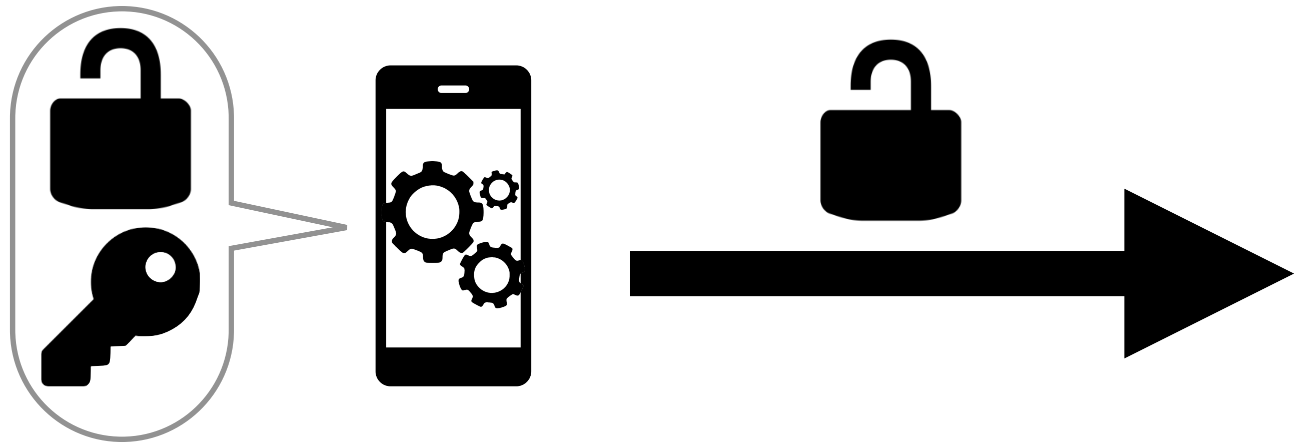
- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio

A Closer Look at Secure Communication



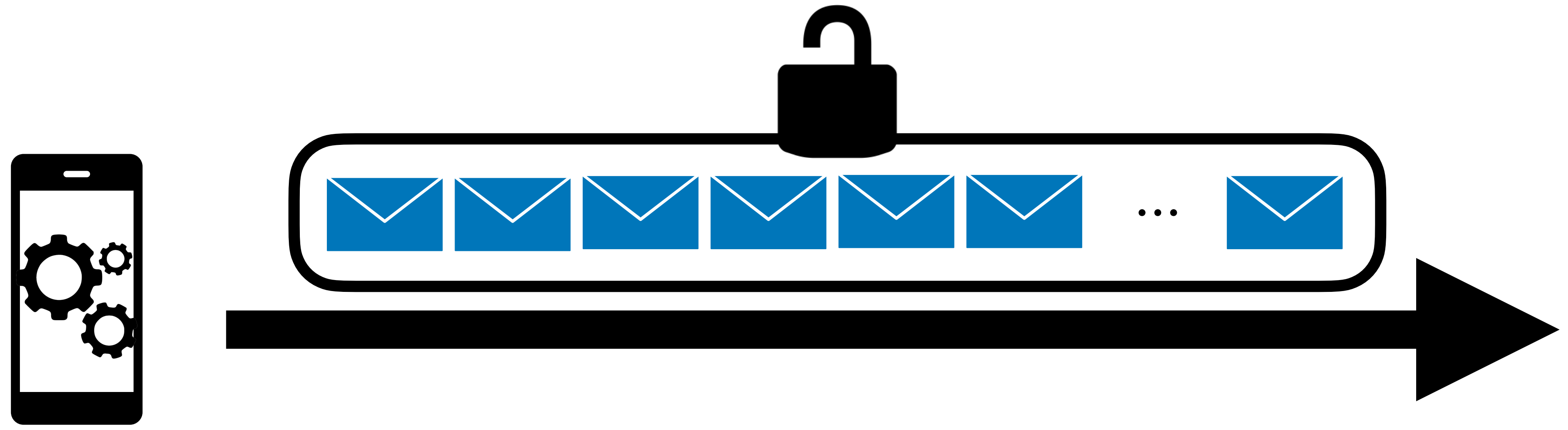
Two Phases:

Key exchange phase



- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot$ |🔒| (not n^2)

Encryption phase



- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio

Back to the PCG Template Again

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathrm{PRG}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}}))$$

Using a PRG enables a *one-time* generation
of a *fixed* amount of correlations

Back to the PCG Template Again

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRF}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

A pseudorandom correlation *function* is to a PCG what a PRF is to a PRG

Back to the PCG Template Again

$$\underbrace{\mathrm{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathrm{PRF}}^{\text{Public function}}(\underbrace{\mathrm{seed}}_{\text{Short seed}}))$$

A pseudorandom correlation *function* is to a PCG what a PRF is to a PRG

Are there any FSS-friendly PRFs?

Back to the PCG Template Again

$$\underbrace{\mathsf{FSS}}_{\text{Function secret sharing}}(\overbrace{C \circ \mathsf{PRF}}^{\text{Public function}}(\underbrace{\mathsf{seed}}_{\text{Short seed}}))$$

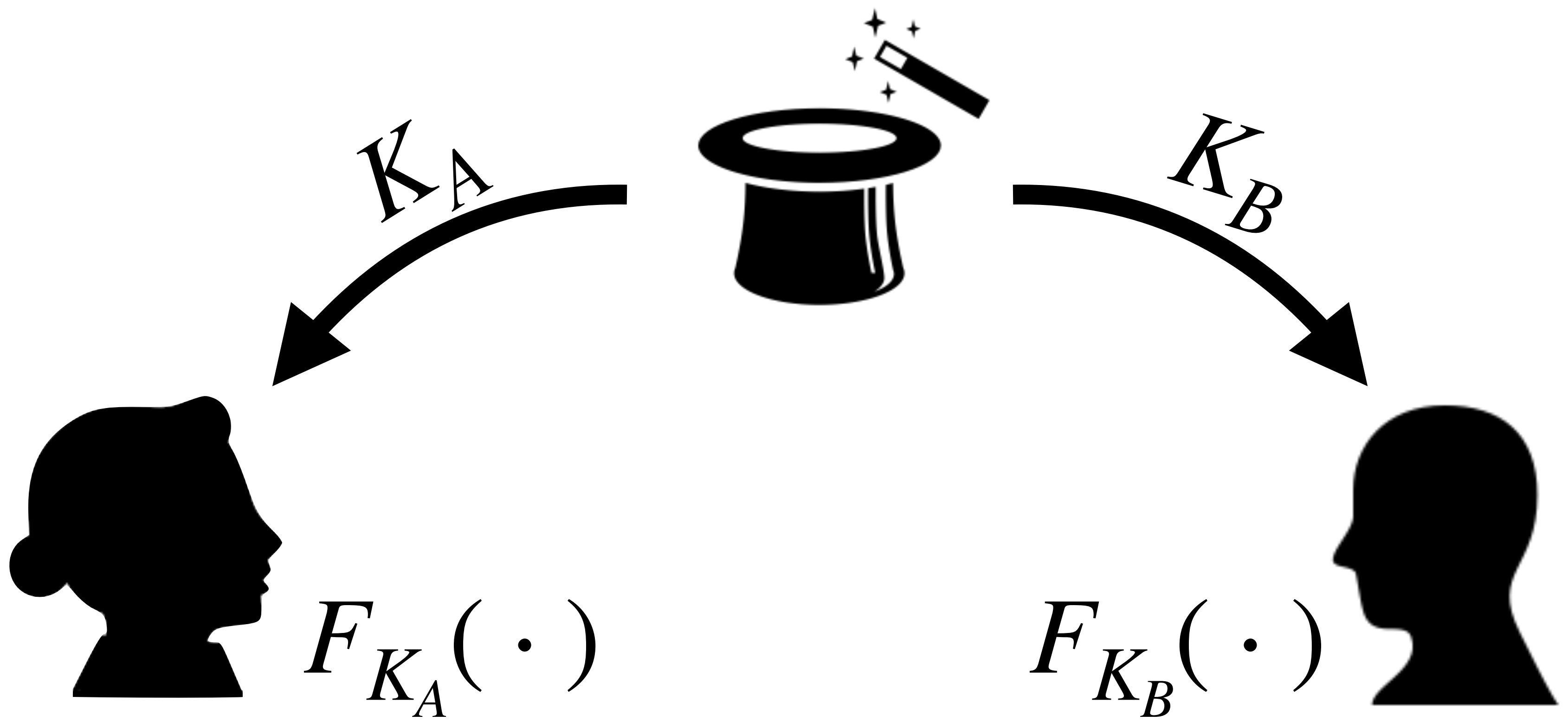
A pseudorandom correlation *function* is to a PCG what a PRF is to a PRG

The existence of PRFs in low complexity classes yields strong limitations for learning theory

Are there any FSS-friendly PRFs?

FOCS:BCGIKS20 and Crypto:BCGIKRS22 give plausible candidates

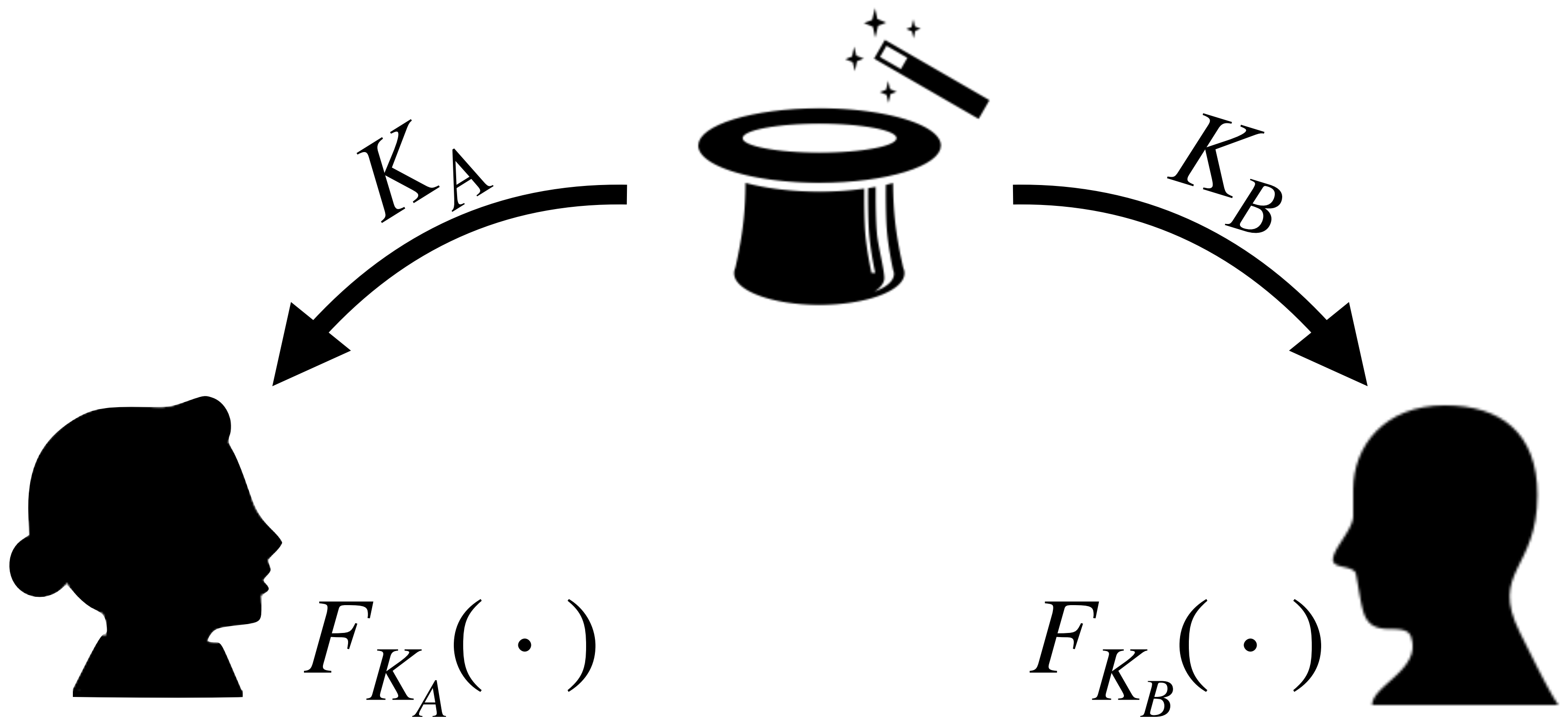
Pseudorandom Correlation Functions



Correctness & security:

- Black-box access to samples of the form $(F_{K_A}(x), F_{K_B}(x))$ are indistinguishable from black-box access to random samples from a target correlation.
- From the viewpoint of Alice, each $F_{K_B}(x)$ is indistinguishable from a random value sampled *conditioned on satisfying the correlation with* $F_{K_A}(x)$.
- Same condition in the other direction.

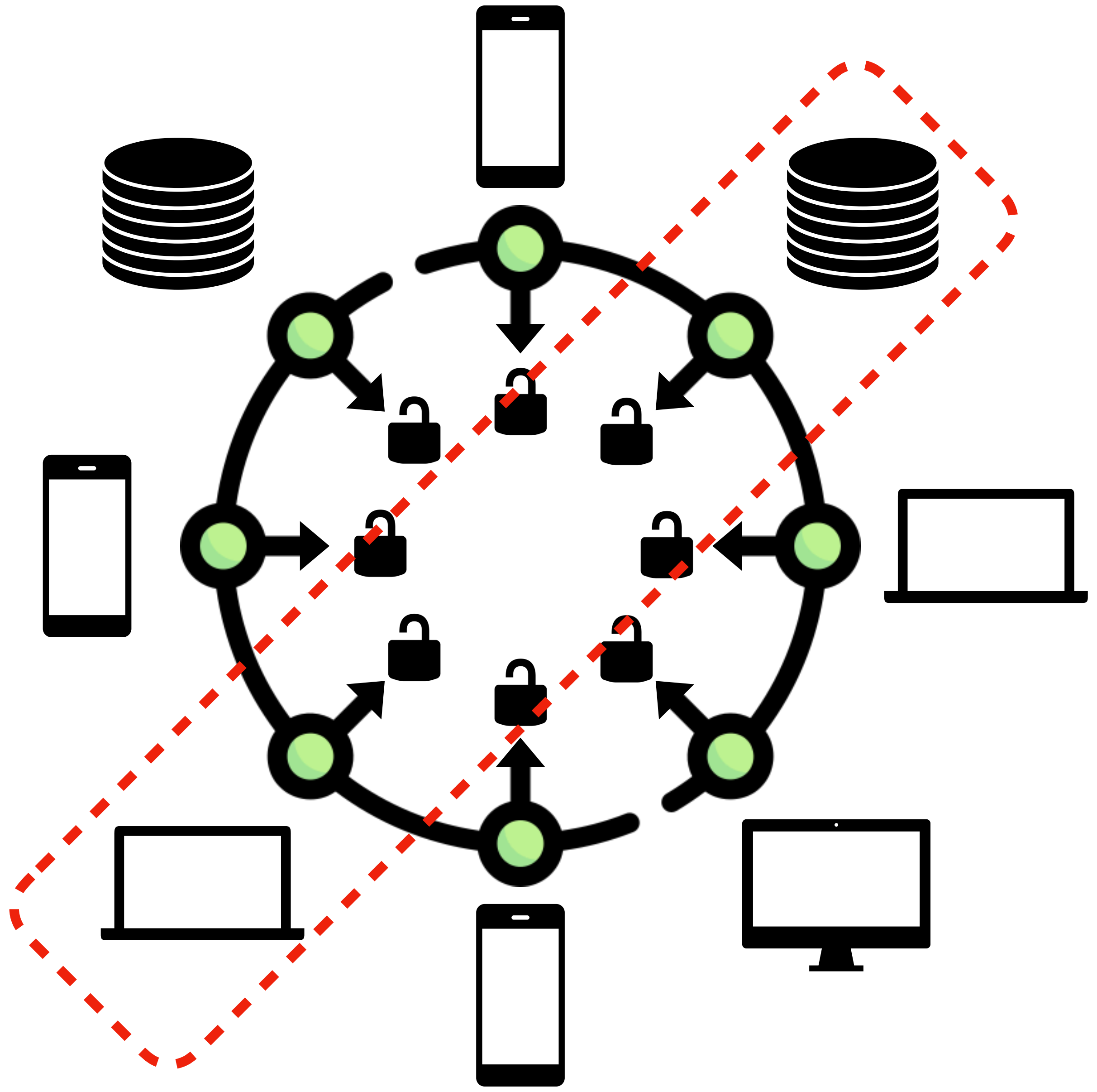
Public-Key Pseudorandom Correlation Functions



Correctness & security:

- Black-box access to samples of the form $(F_{K_A}(x), F_{K_B}(x))$ are indistinguishable from black-box access to random samples from a target correlation.
- From the viewpoint of Alice, each $F_{K_B}(x)$ is indistinguishable from a random value sampled *conditioned on satisfying the correlation with* $F_{K_A}(x)$.
- Same condition in the other direction.

Achieving *non-interactive* silent key generation



Formally:

- KeyGen → (pk, sk) generates public and private PCF keys
- KeyDer(pk$_A$, sk$_B$) → K_B^{AB} yields Bob's PCF key w.r.t. Alice's key
- Eval$(K, x) \rightarrow y$ yields a pseudorandom sample

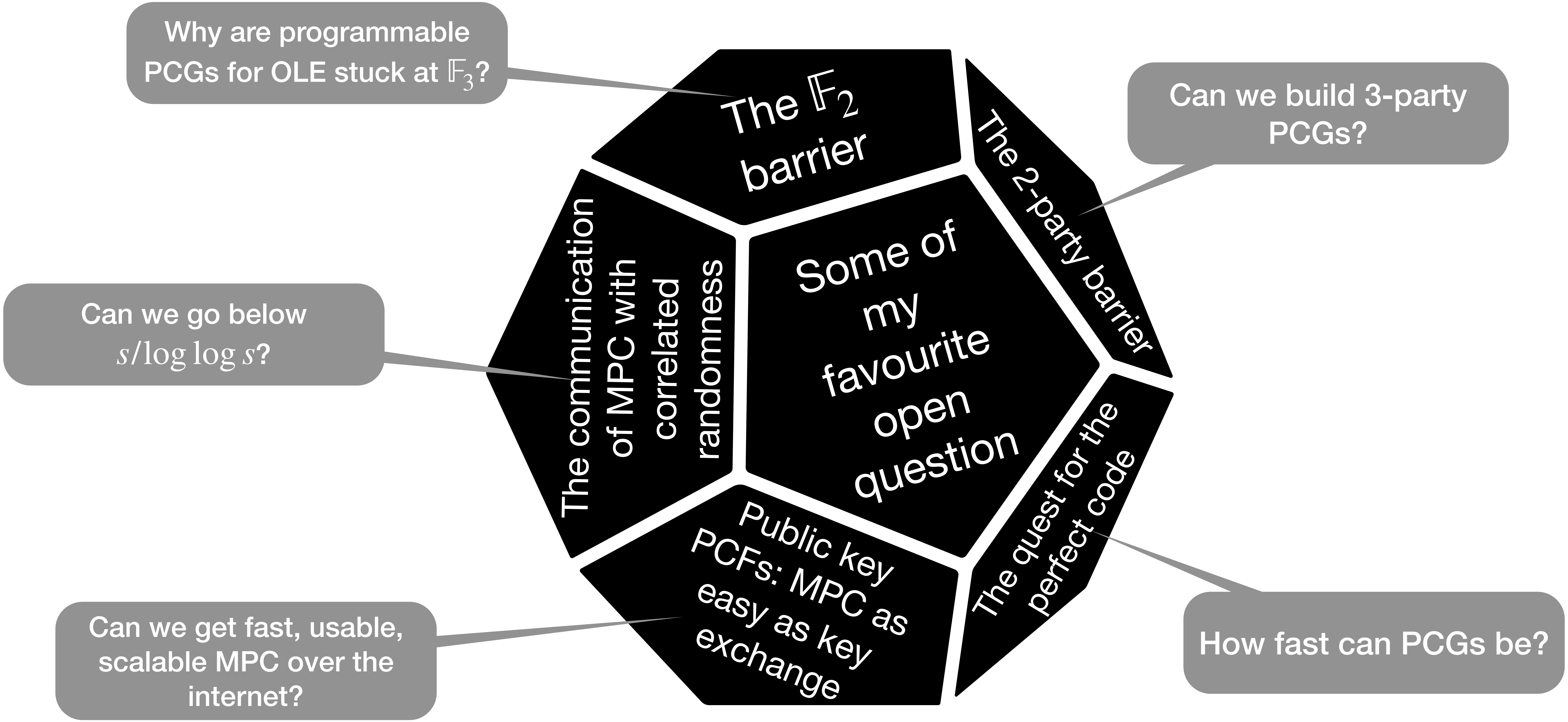
Public-Key Pseudorandom Correlation Functions

Public-key PCFs are exactly the *right tool* to enable scalable, on-demand 2-party secure computation over the Internet, with a communication and computation pattern close to that of secure *communication* over the web.

Building efficient public-key PCF is essentially a wide-open question: the recent work of EC:Orlandi-Scholl-Yakoubov'21 gets it for OT from QR, but efficiency is quite bad.

(Teaser) Coming soon: we have some exciting progress in this line of work, which does not fully solve the problem, but is a big step forward!

Some of my Favourite Open Questions



Some of my favourite open question

The $\mathbb{F}_2$ barrier

Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The 2-party barrier

Can we build 3-party PCGs?

The communication of MPC with correlated randomness

Can we go below $s/\log\log s$?

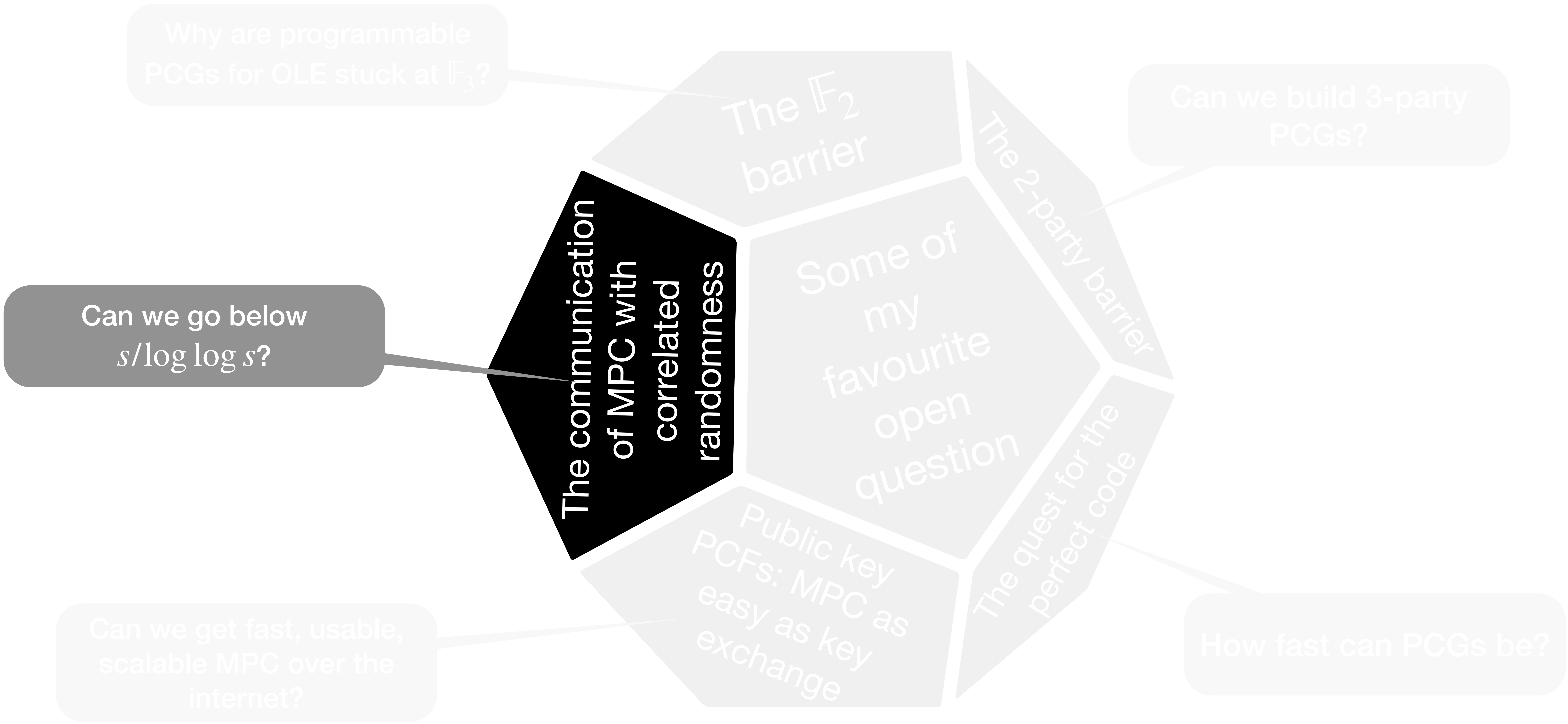
Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The quest for the perfect code

How fast can PCGs be?

Some of my Favourite Open Questions



Why are programmable PCGs for OLE stuck at $\mathbb{F}_3$?

The $\mathbb{F}_2$ barrier

Can we build 3-party PCGs?

The 2-party barrier

Can we go below $s/\log\log s$?

The communication of MPC with correlated randomness

Some of my favourite open question

Public key PCFs: MPC as easy as key exchange

Can we get fast, usable, scalable MPC over the internet?

The quest for the perfect code

How fast can PCGs be?

Some of my Favourite Open Questions

Can we go below $s/\log\log s$?

No time left for that, but I'd be happy to discuss it over dinner tonight!

Other cool things to check out that I don't have time to discuss:

- People have been doing great things in zero-knowledge using these PCG techniques (incl. right here in Aarhus!)
- Everything we have so far works only for two parties!
- … And many more

Questions?