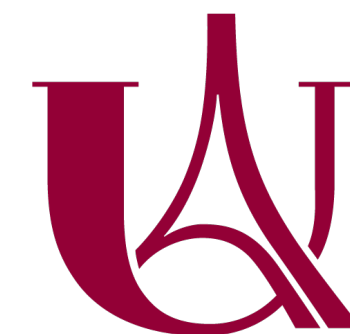


Correlated Pseudorandomness

Achieving faster secure computation through
pseudorandom correlation generators

Geoffroy Couteau



Université
de Paris

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...

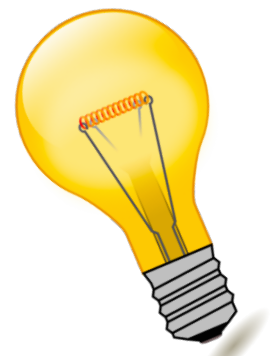


Start of the Story: Circa 2016



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...



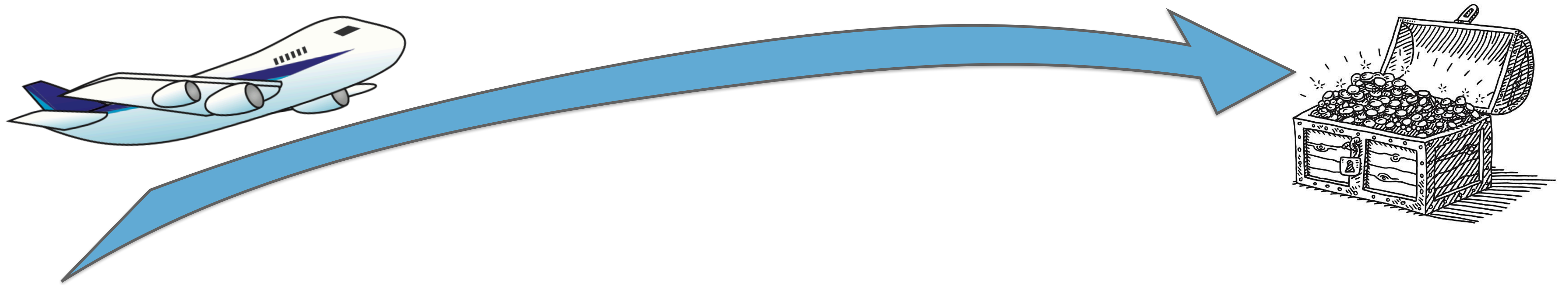
A few months and a CCS’17 paper later, we concluded that the answer was ‘not so much’



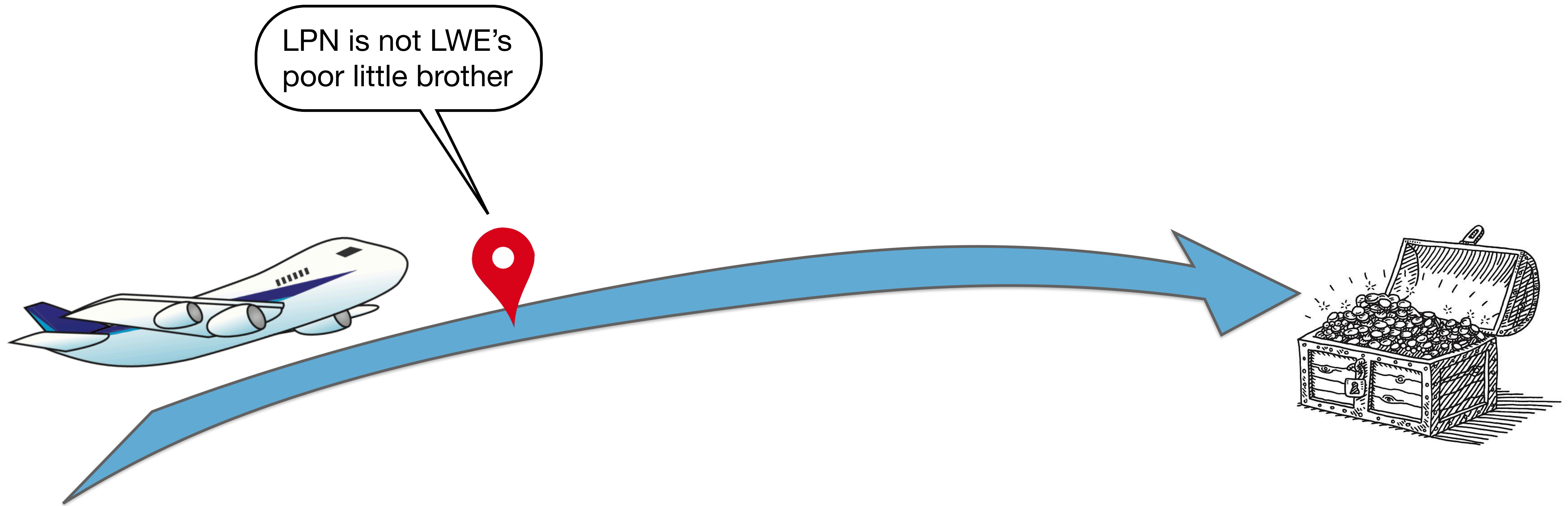
7 years later



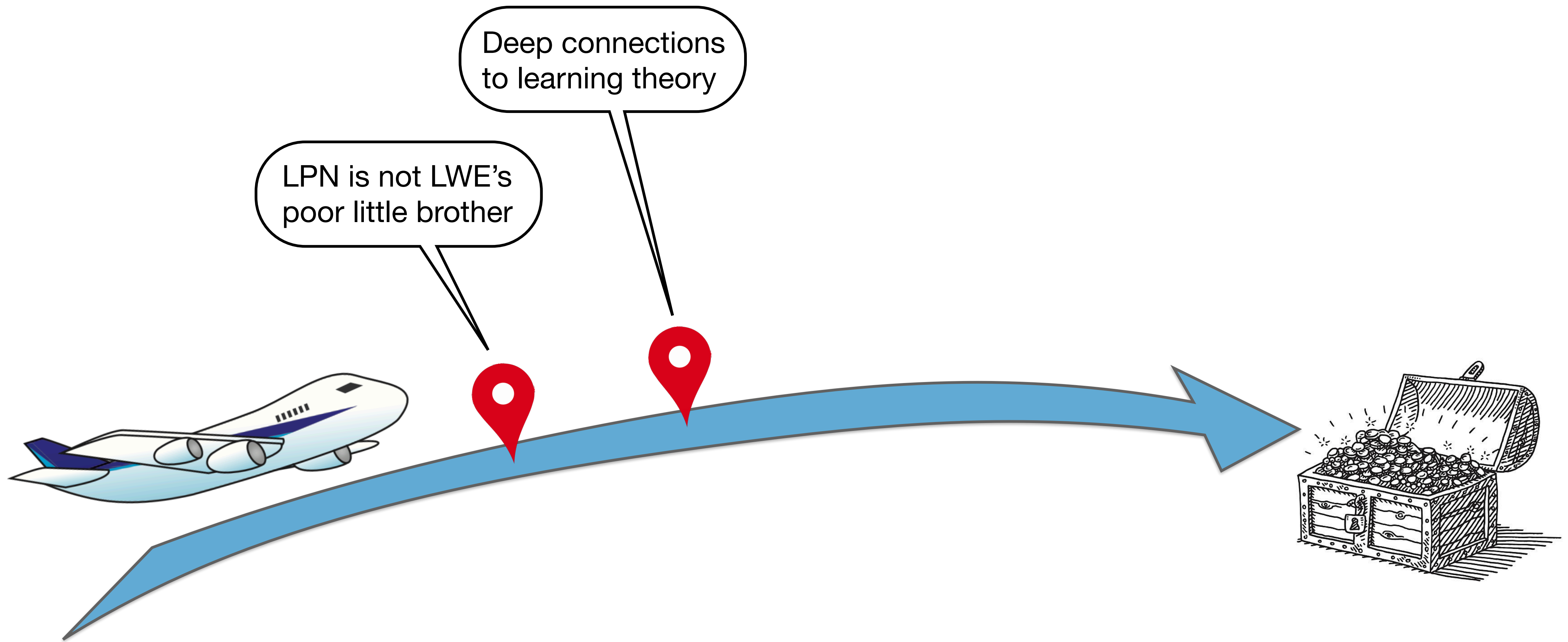
The Journey through Correlated Pseudorandomness



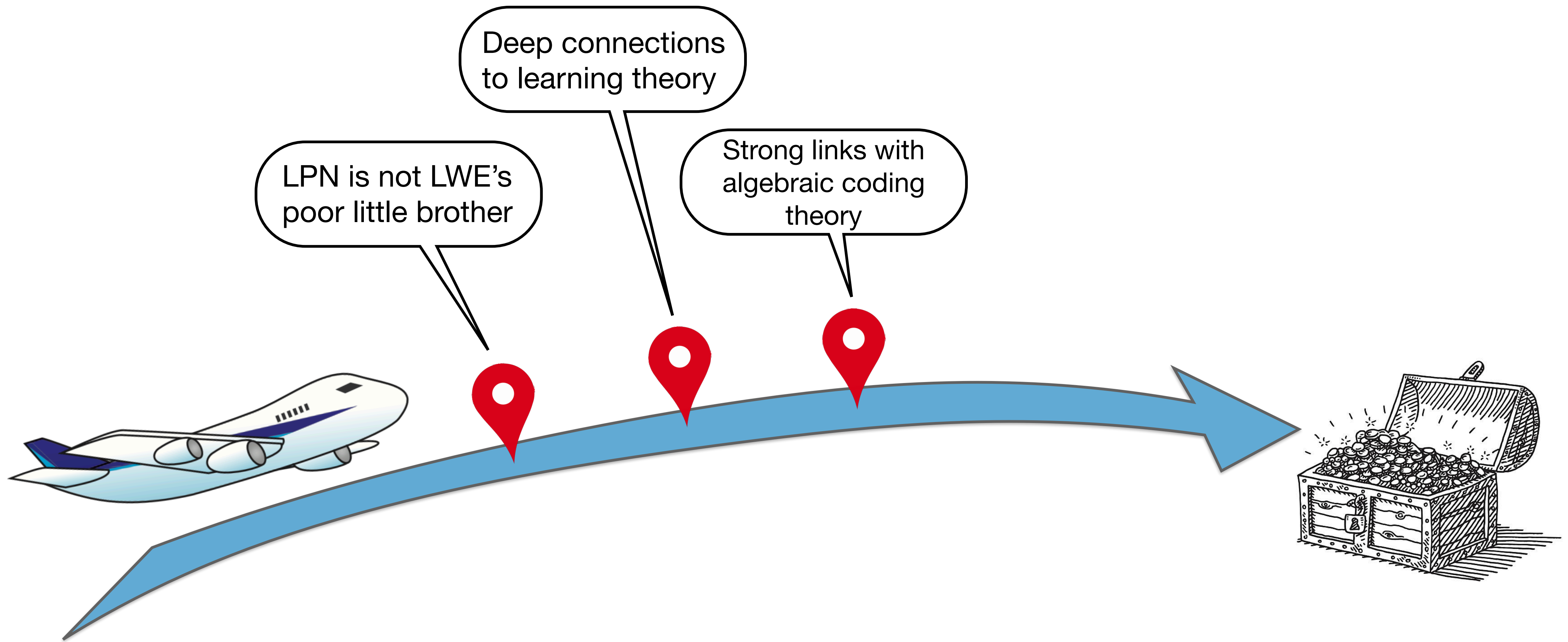
The Journey through Correlated Pseudorandomness



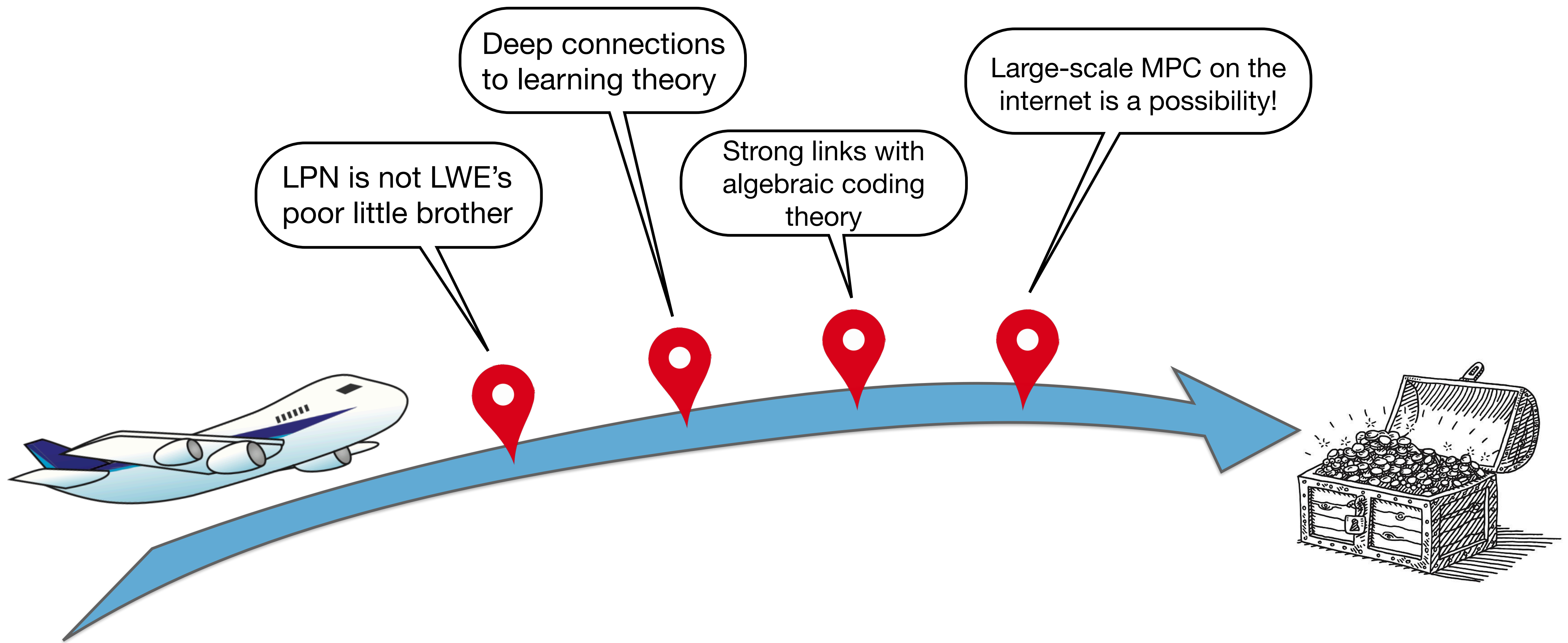
The Journey through Correlated Pseudorandomness



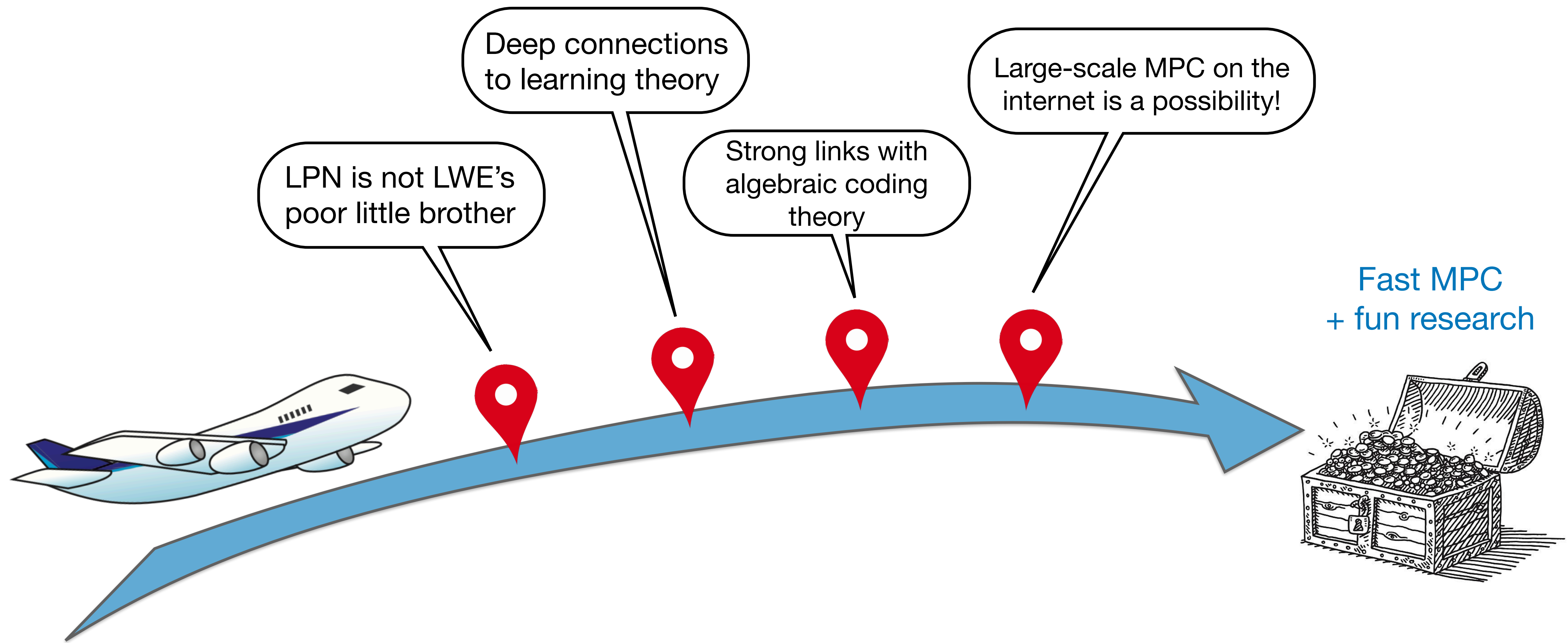
The Journey through Correlated Pseudorandomness



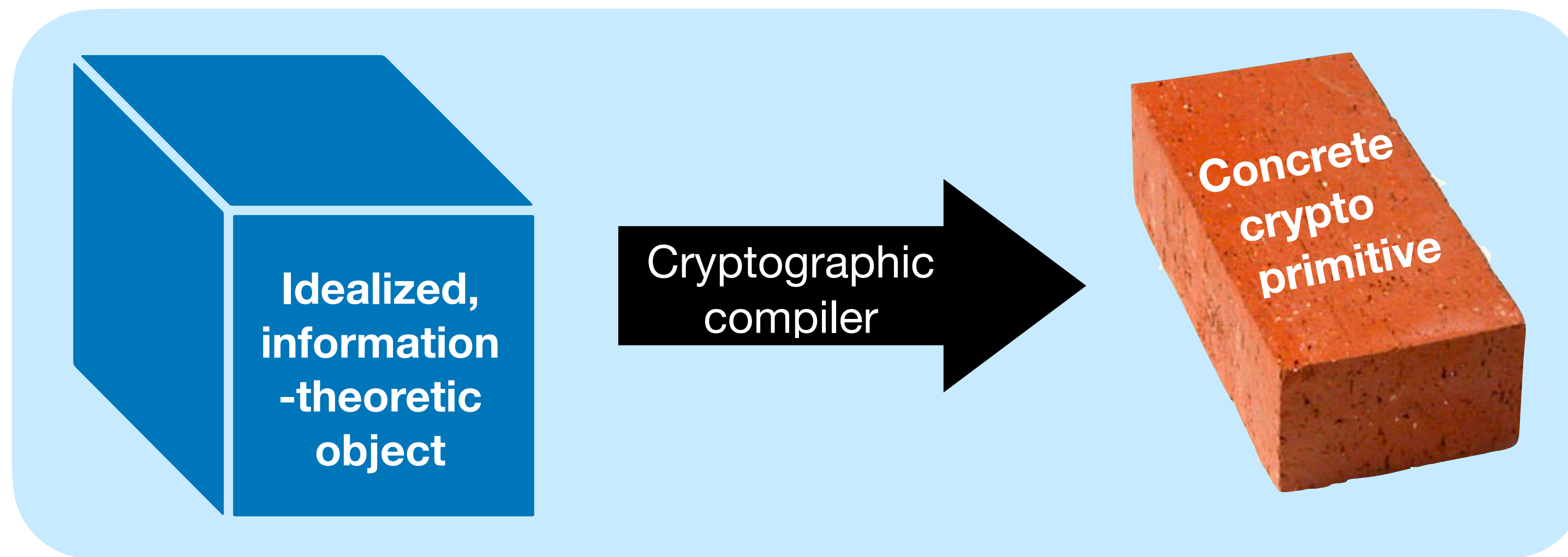
The Journey through Correlated Pseudorandomness



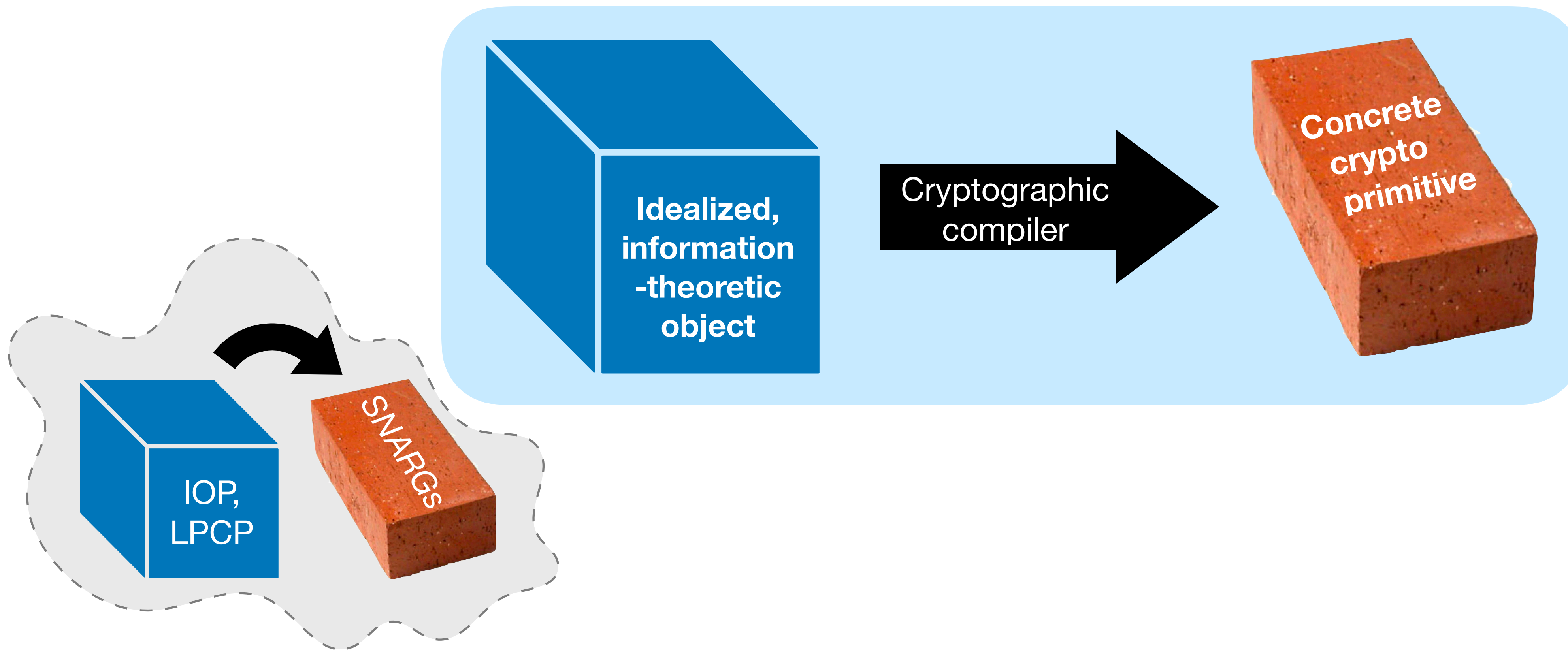
The Journey through Correlated Pseudorandomness



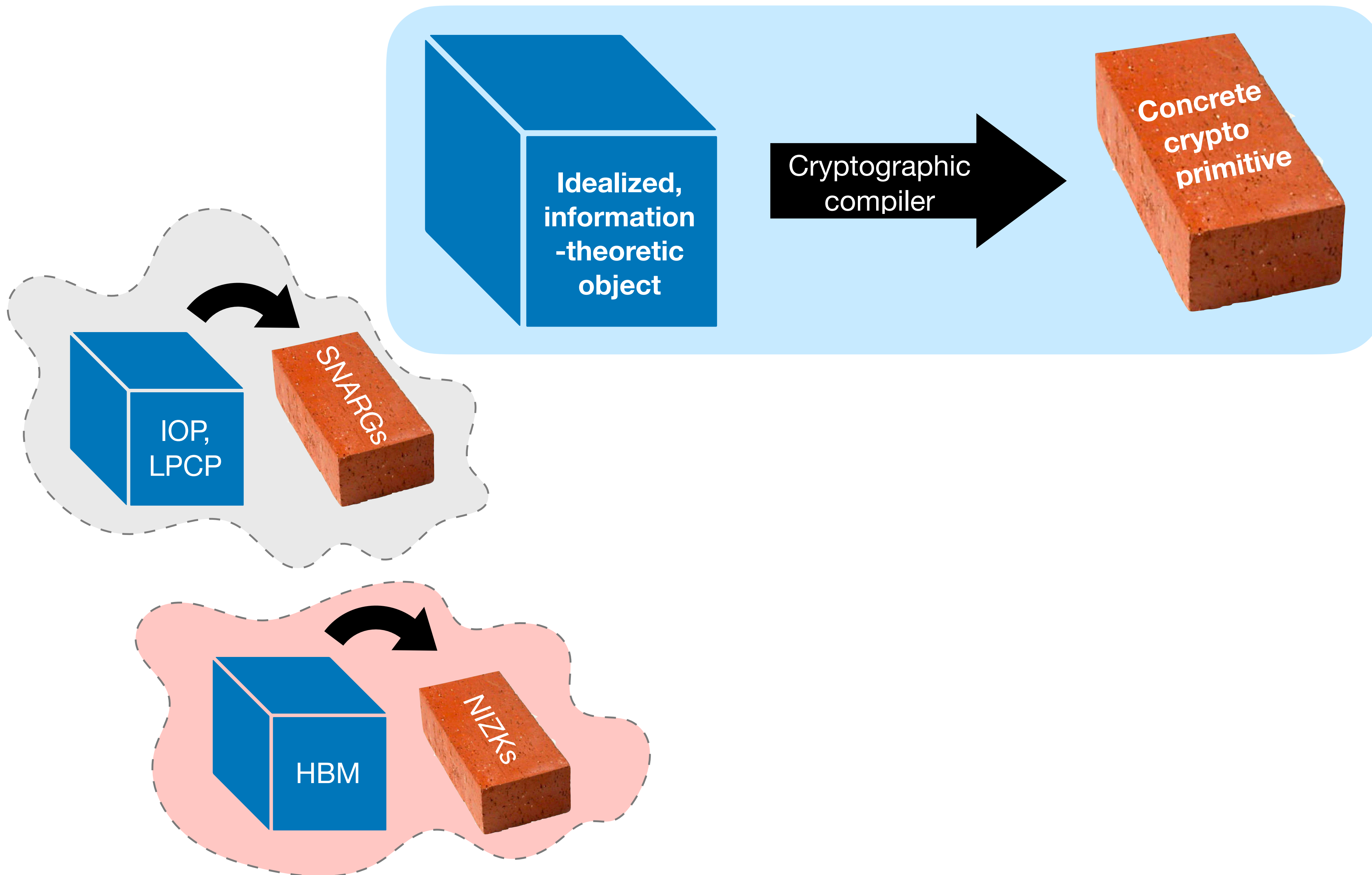
A Standard Cryptographic Approach



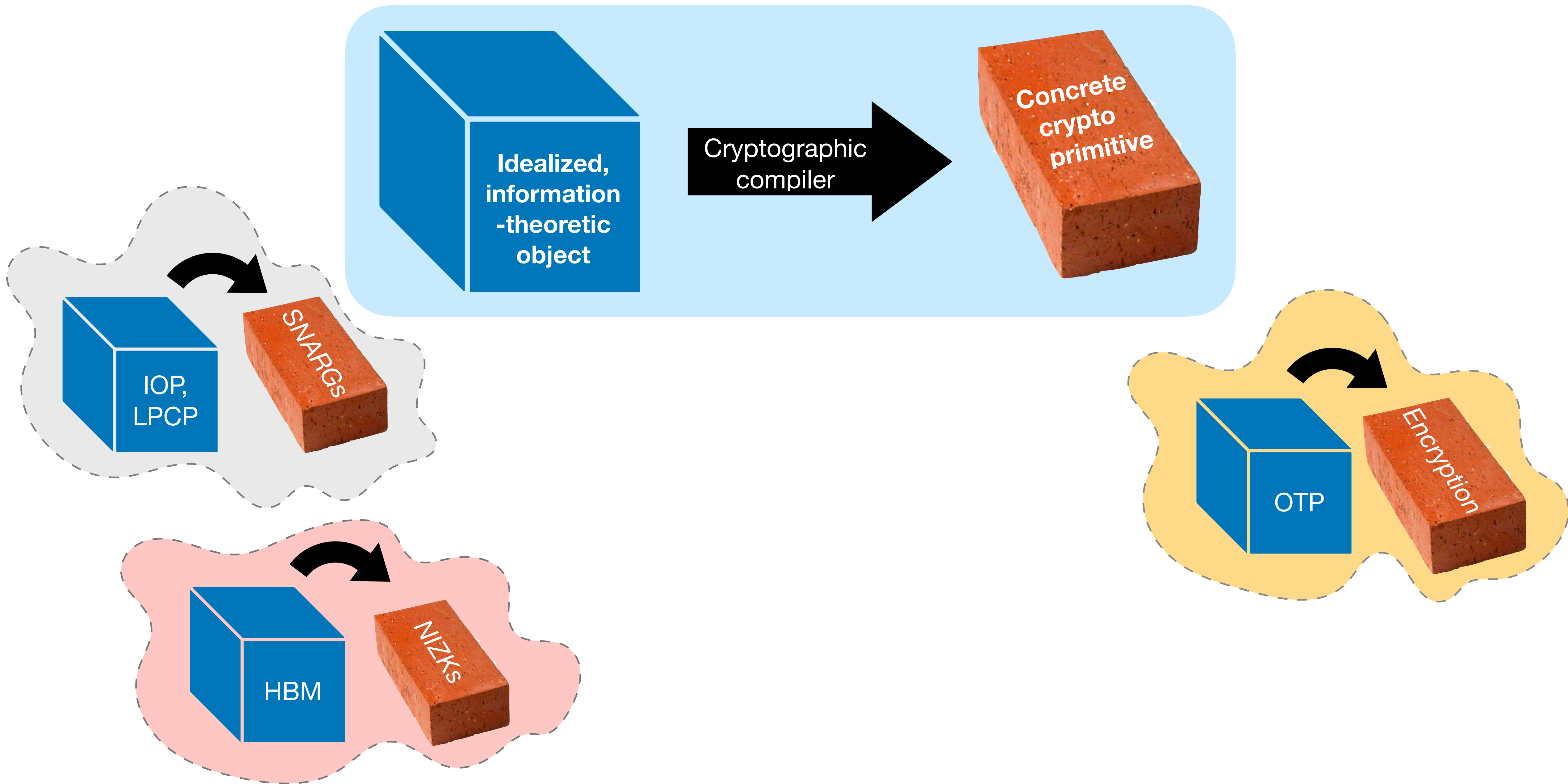
A Standard Cryptographic Approach



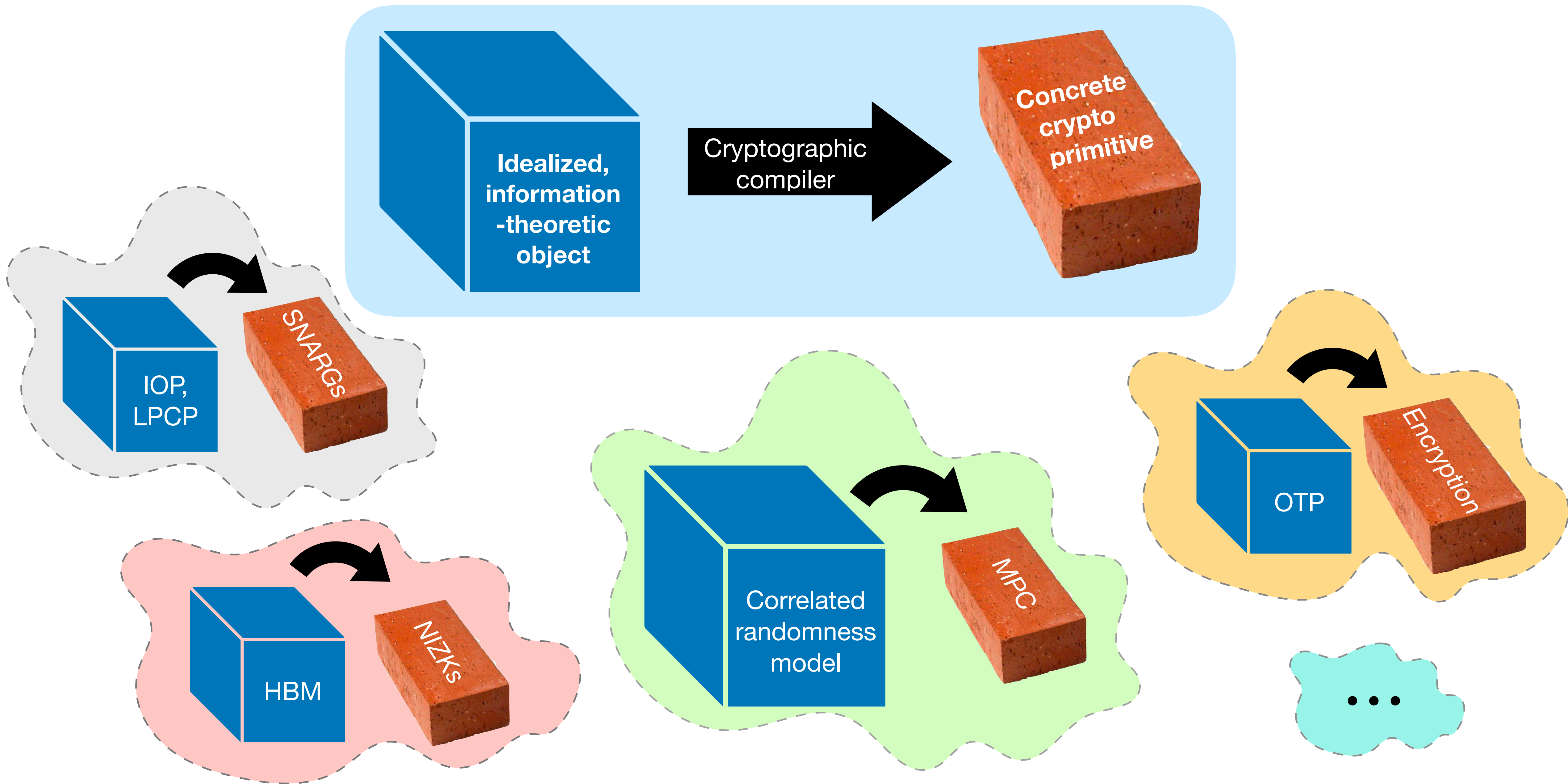
A Standard Cryptographic Approach



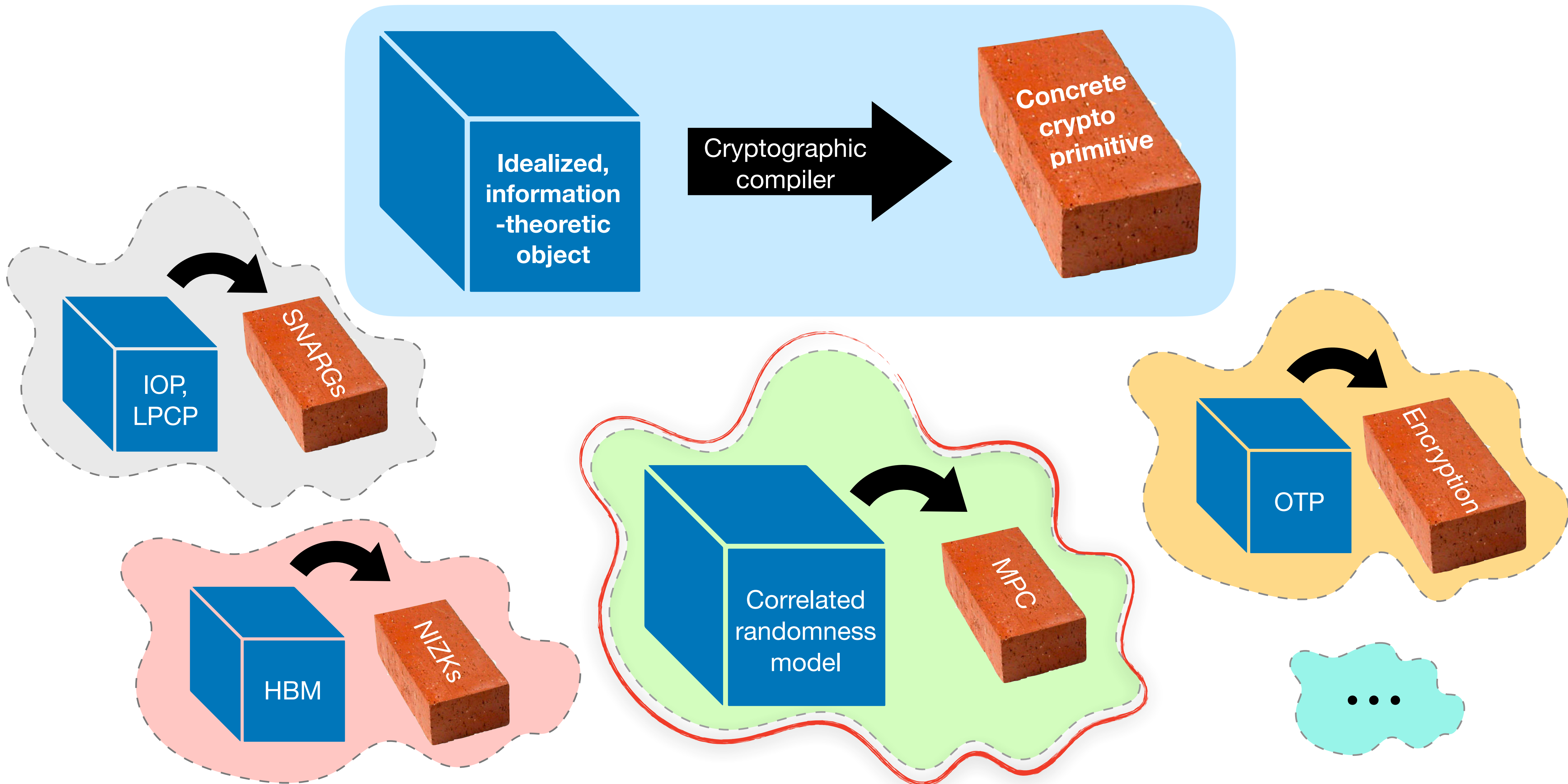
A Standard Cryptographic Approach



A Standard Cryptographic Approach

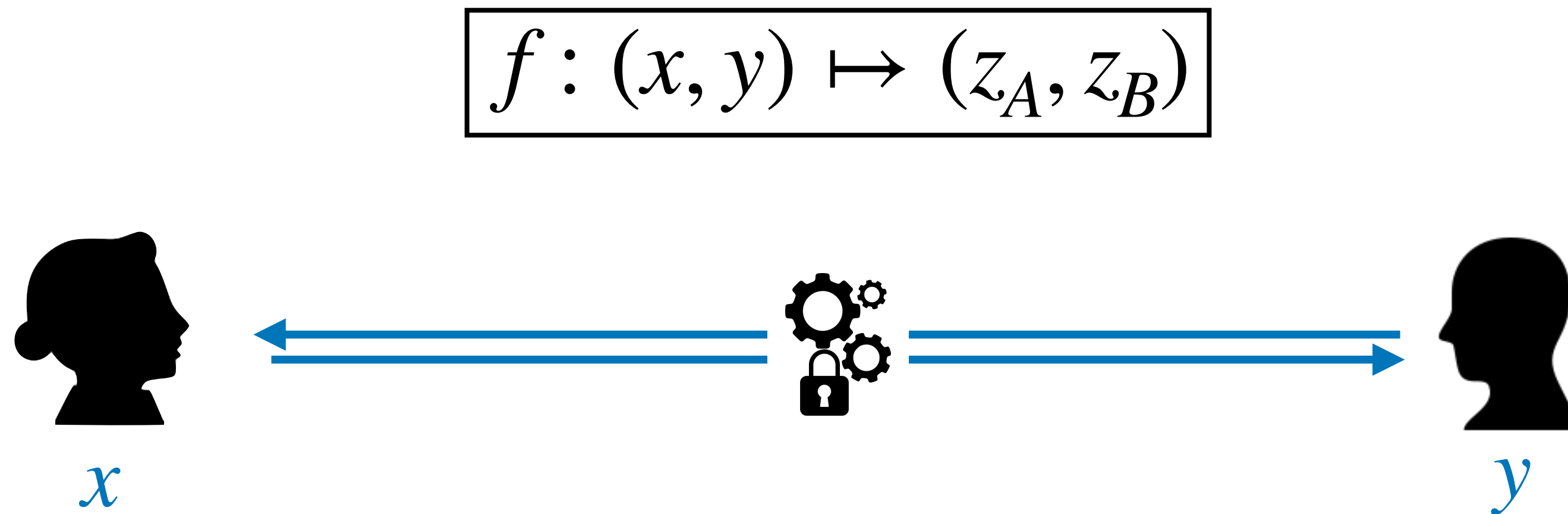


A Standard Cryptographic Approach



Secure Computation...

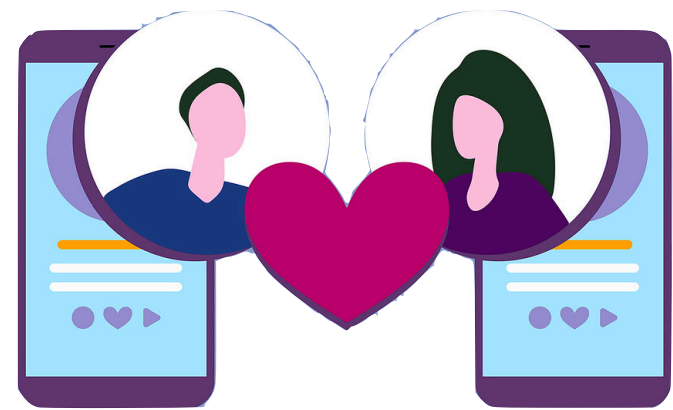
- **Goal.** Computing a **public** function on secret inputs
- **Model.** n players, each with a private input x_i interacting through authenticated channels



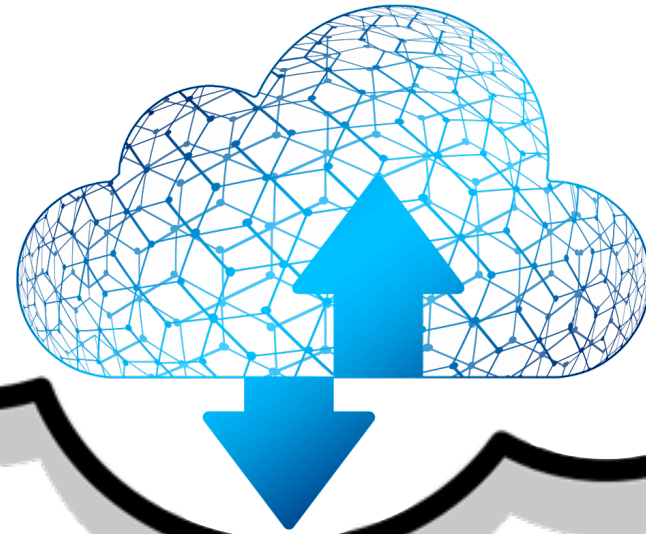
- **Output:** Alice learns z_A and Bob learn z_B
- **Security:** Alice and Bob learn nothing else

... Is a Practical Concern.

Use a dating app



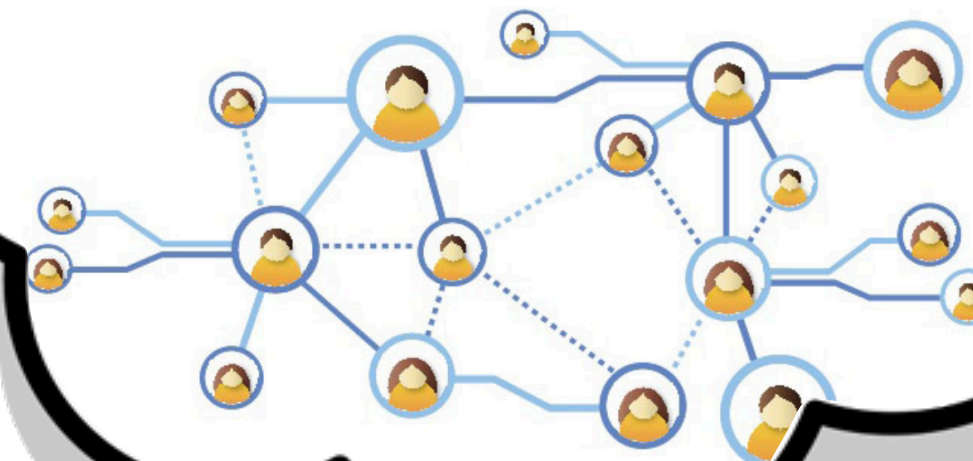
Search over our
Cloud storage



See a targeted
advertising



Use a social network



Get a recommendation
on a streaming platform

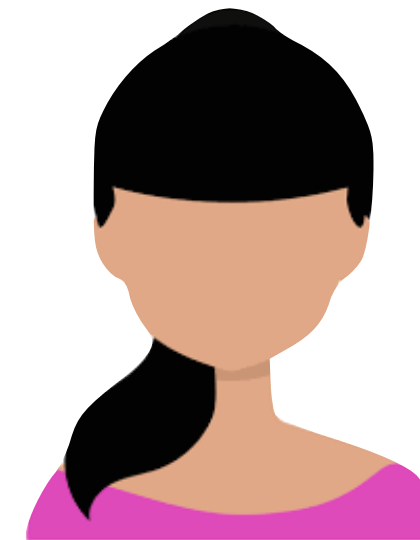
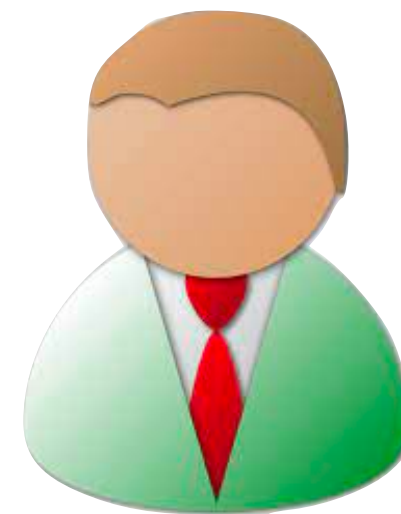
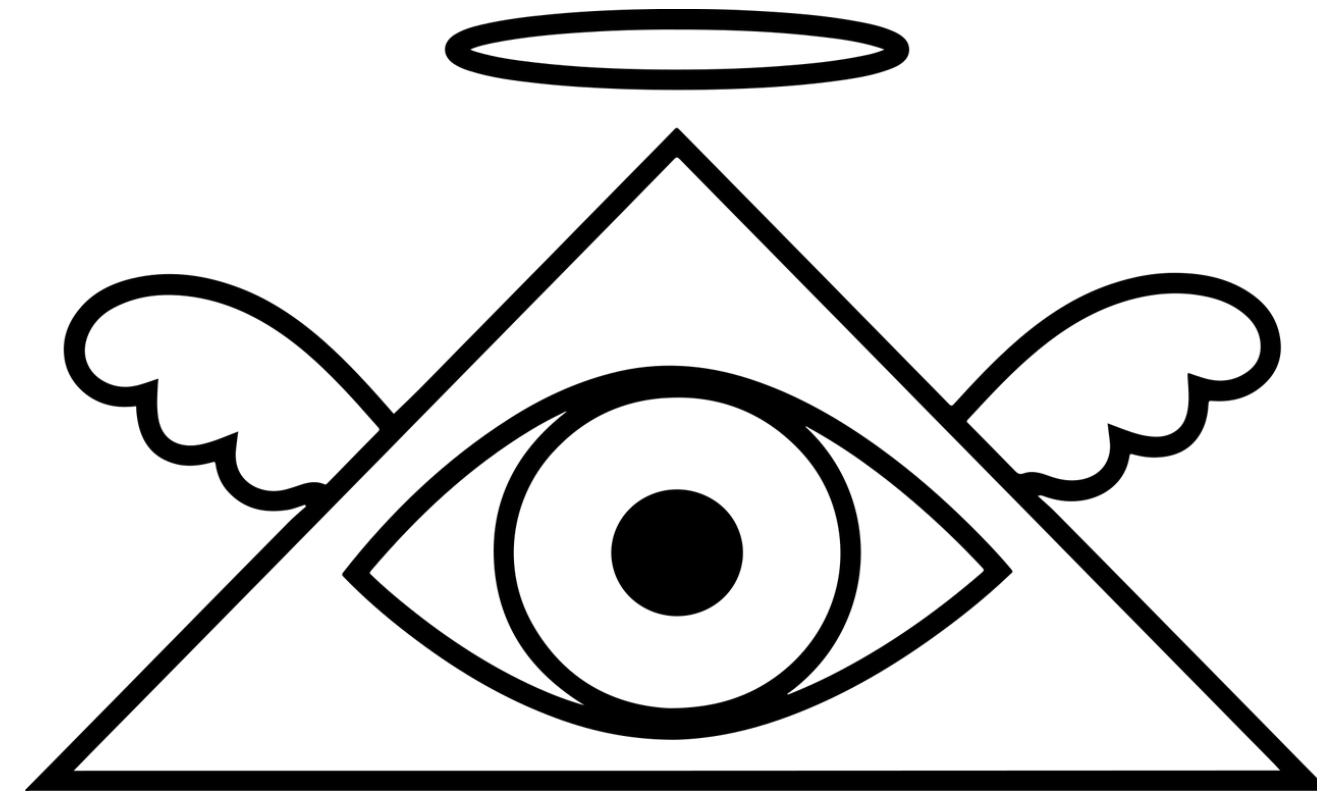


Use a
healthcare
app



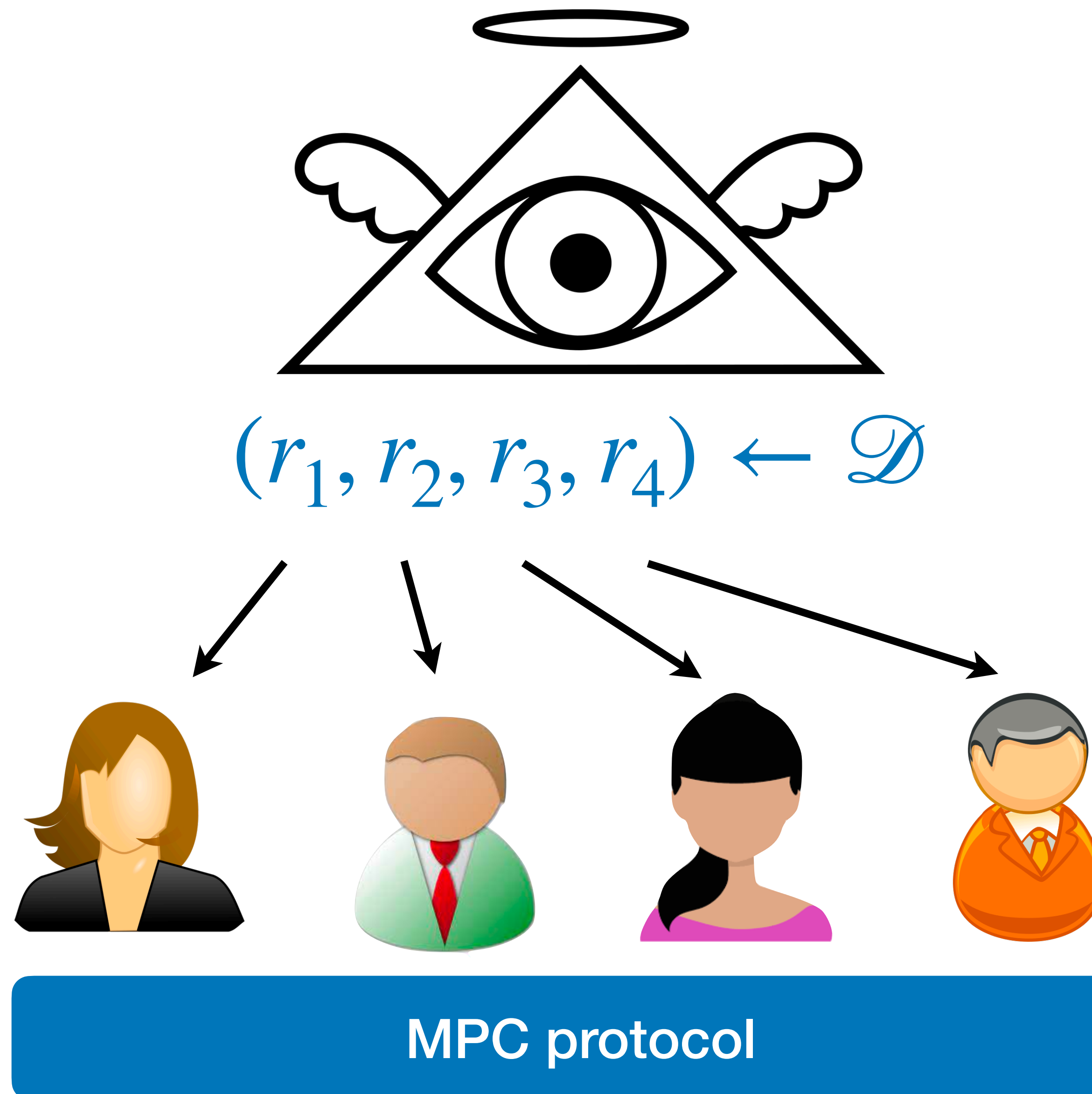
■ ■ ■

The Correlated Randomness Model

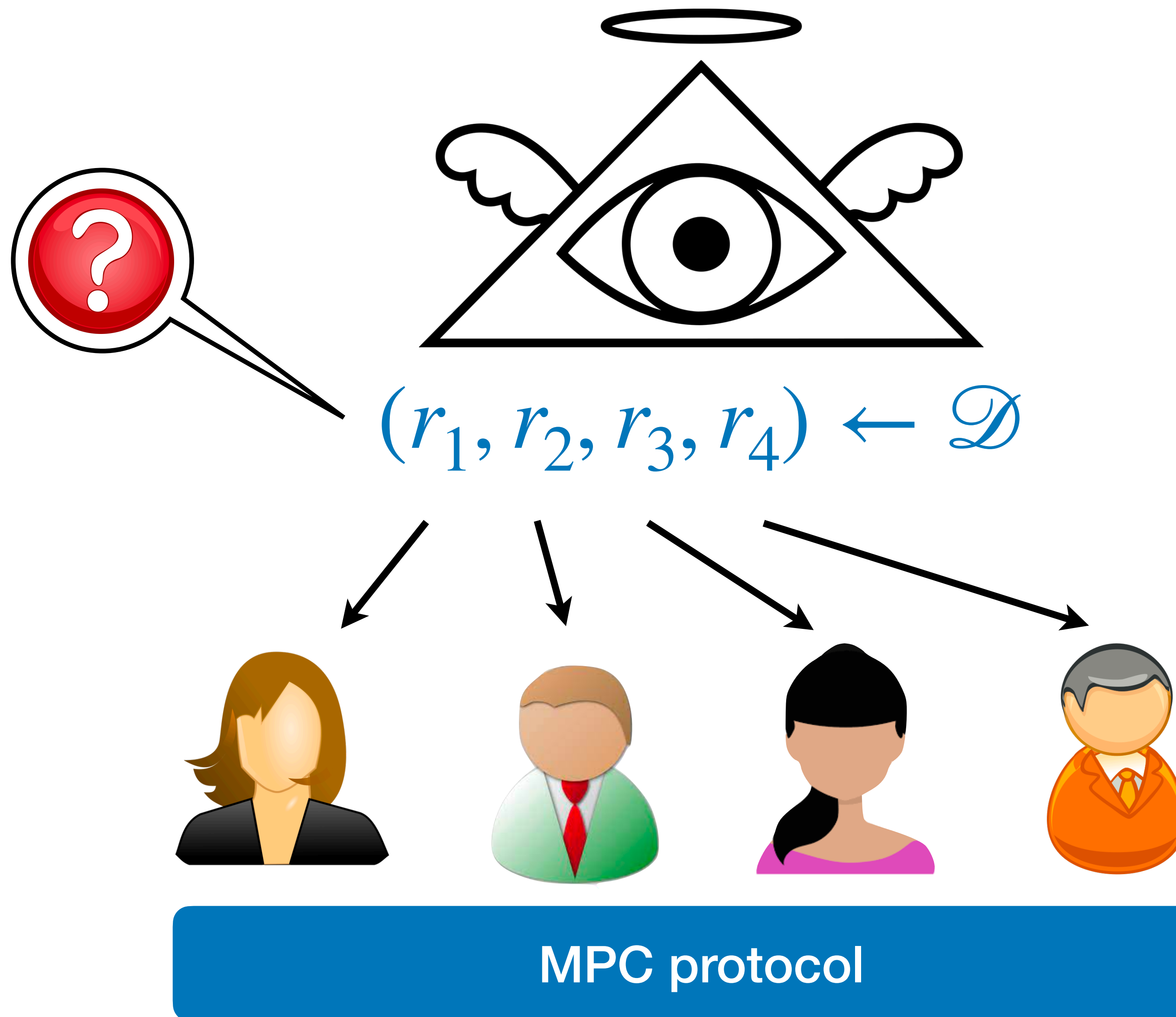


MPC protocol

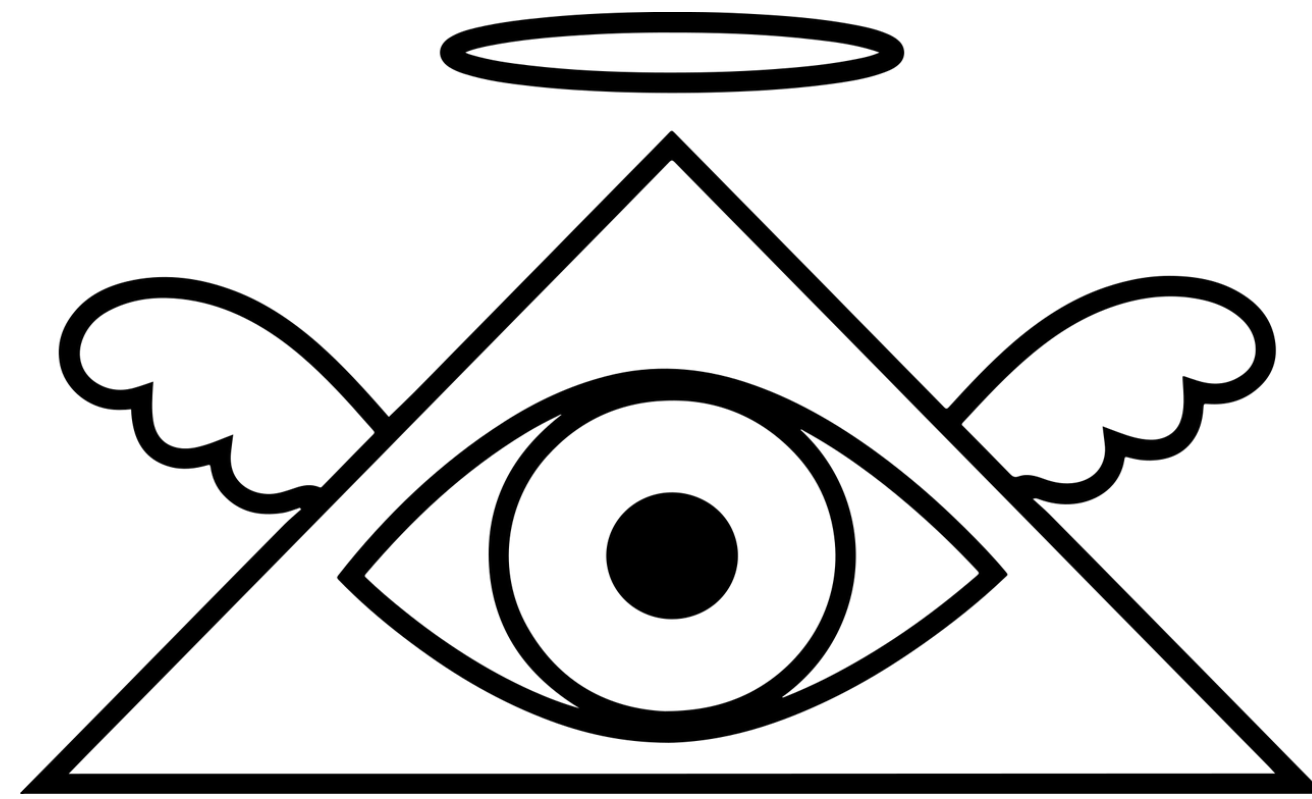
The Correlated Randomness Model



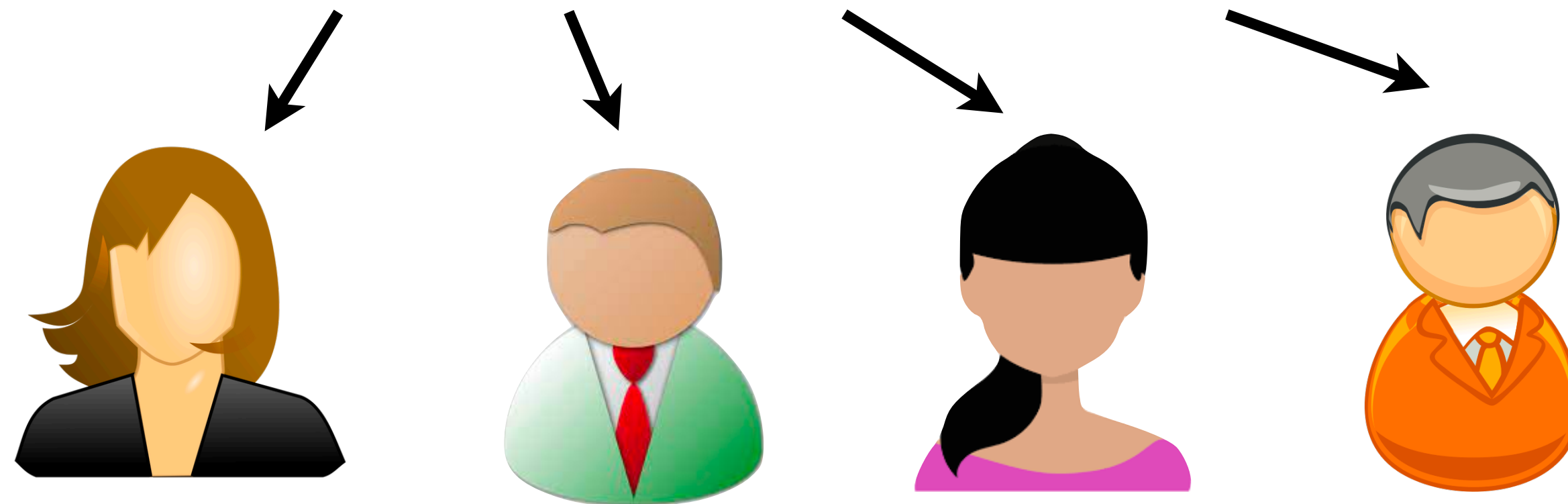
The Correlated Randomness Model



The Correlated Randomness Model

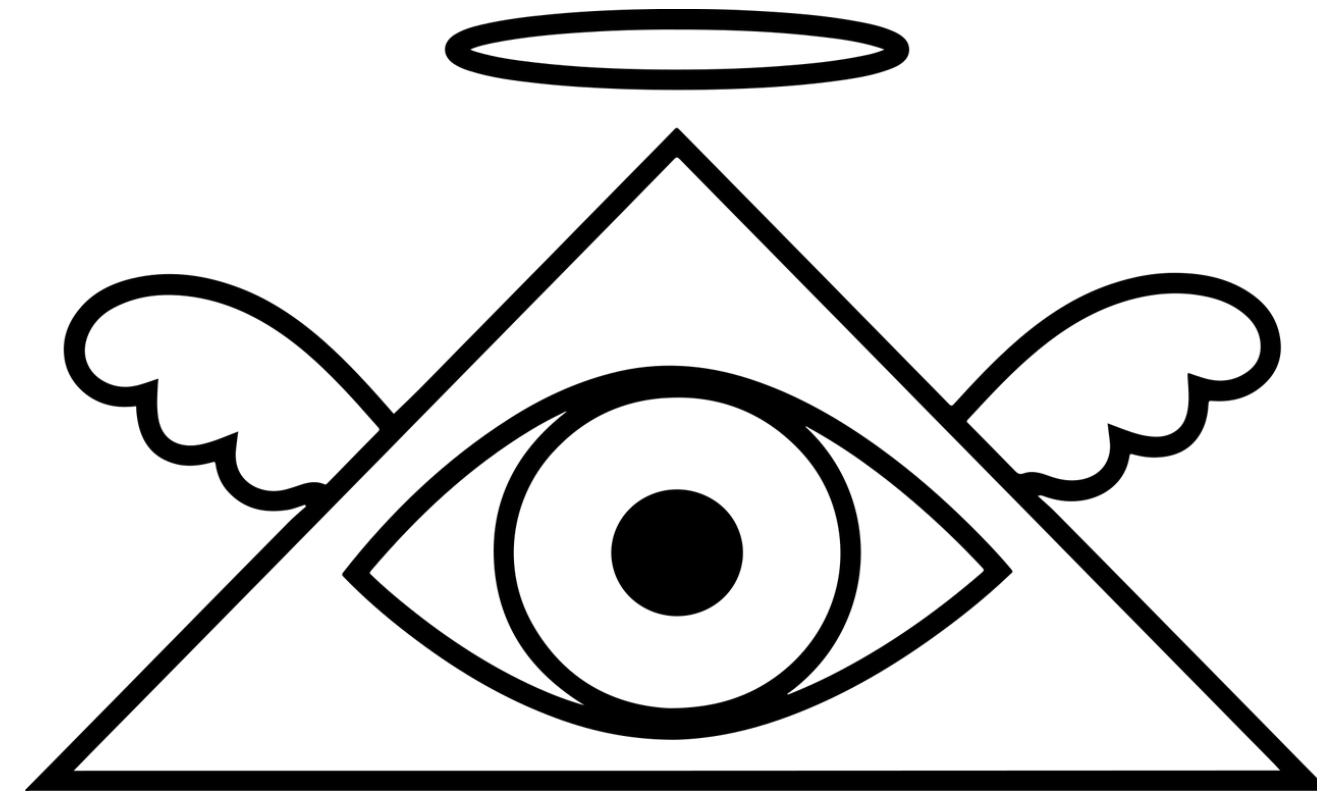


Additive correlations $\left\{ \begin{array}{l} r \leftarrow \mathcal{D} \\ (r_1, r_2, r_3, r_4) \leftarrow \text{shares}(r) \end{array} \right.$



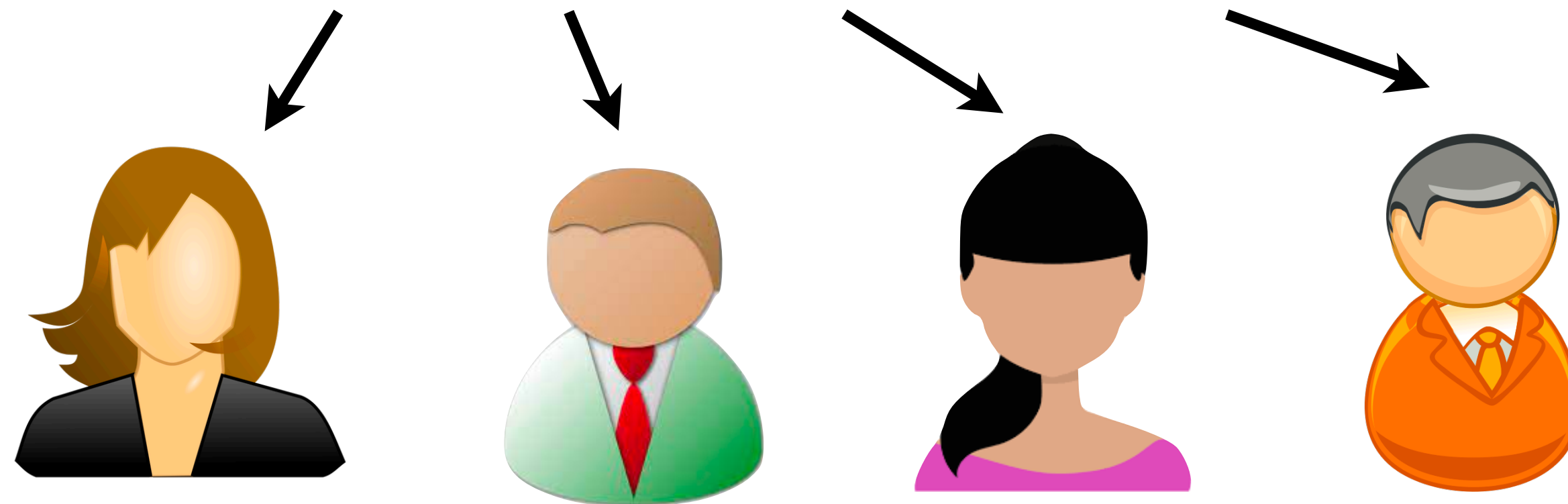
MPC protocol

The Correlated Randomness Model



Example: Beaver triples

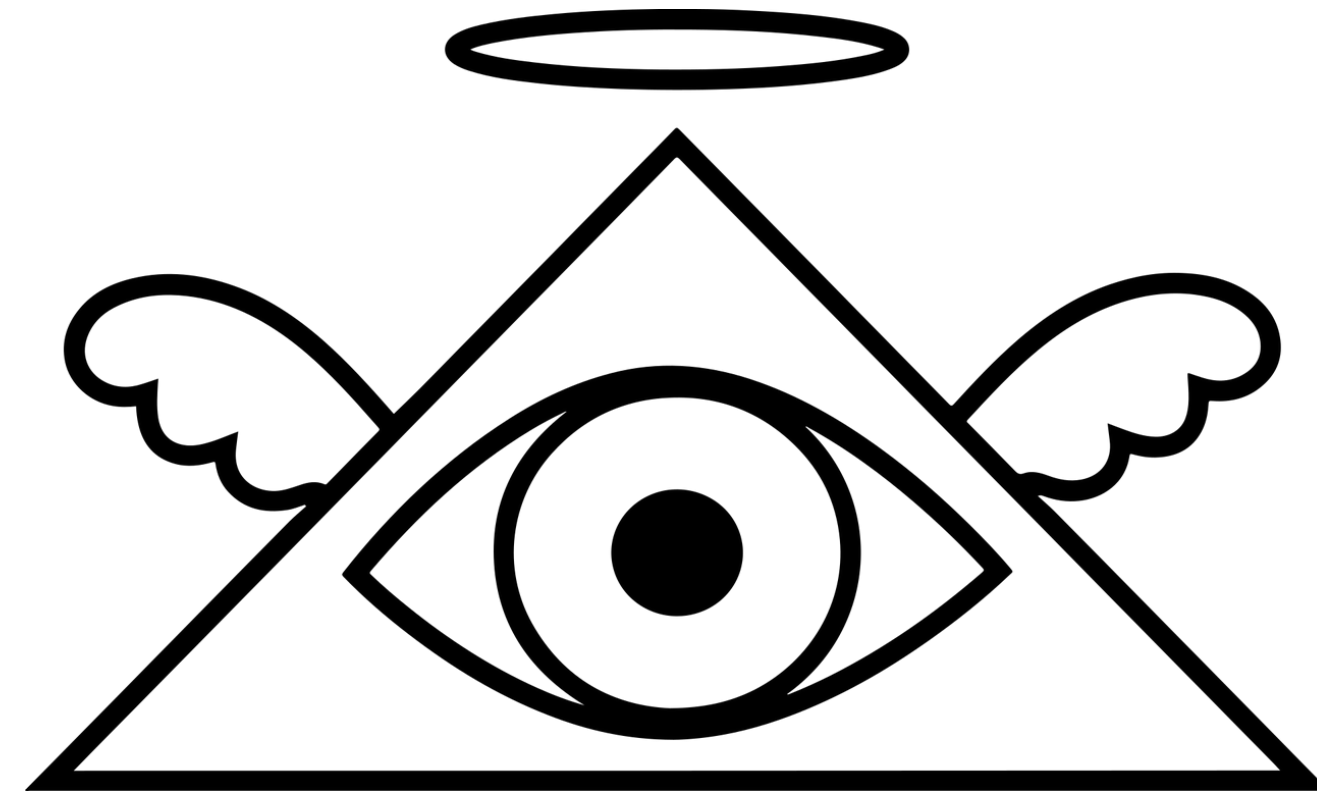
$$\left\{ \begin{array}{l} (a_i, b_i)_{i \leq n} \leftarrow (\mathbb{F}_2 \times \mathbb{F}_2)^n \\ \text{shares}((a_i, b_i, a_i \cdot b_i)_{i \leq n}) \end{array} \right.$$



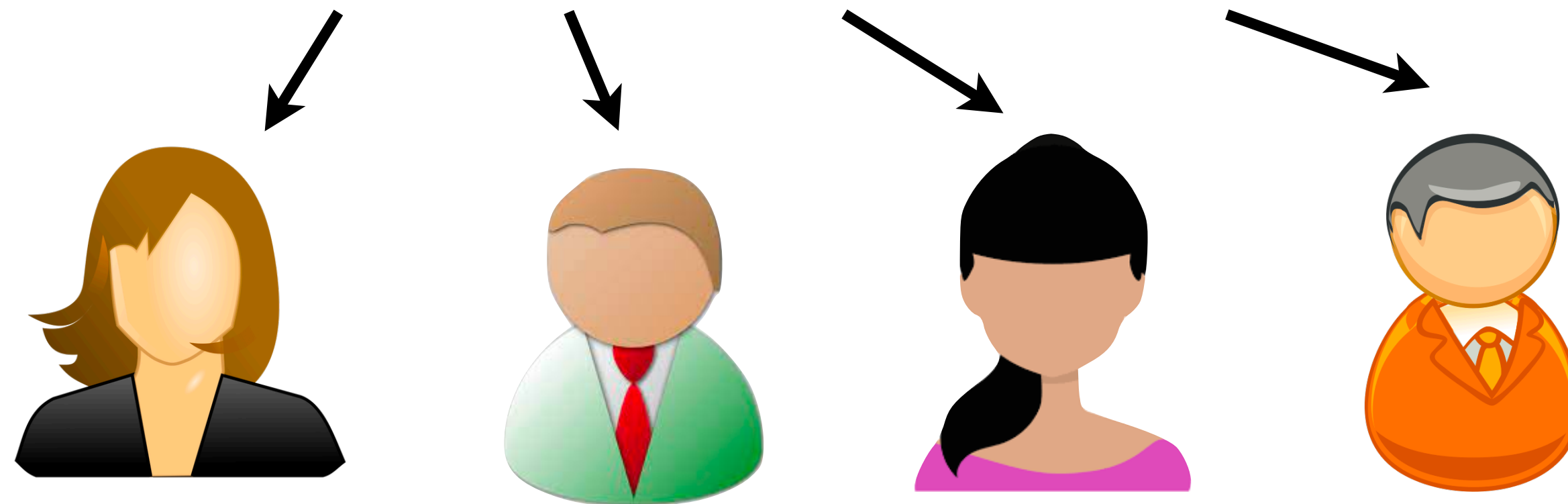
$2N$ bits / $\wedge$ gate
for N parties

GMW protocol

The Correlated Randomness Model



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



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

GMW protocol



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

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

- (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like n samples from the target correlation, and
- (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$ ’ to Bob (similar property w.r.t. Alice).

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

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

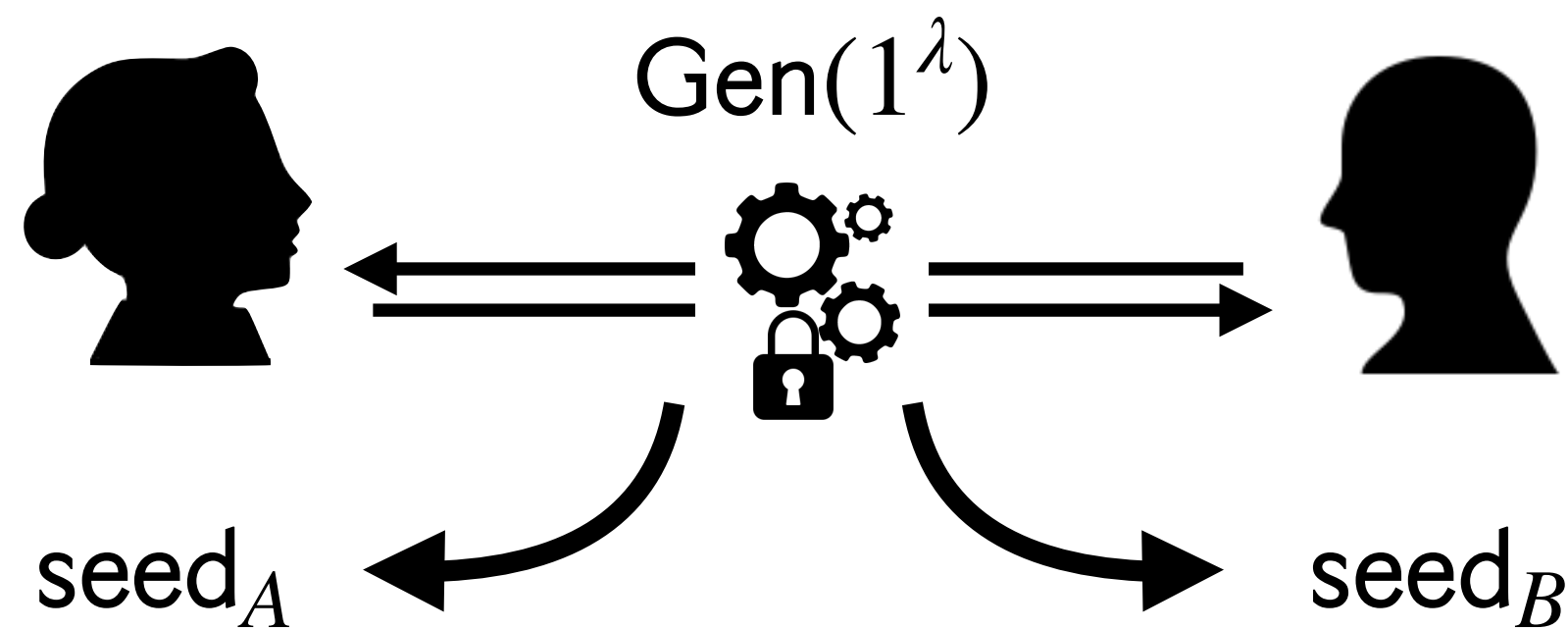
Preprocessing phase

Online phase

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

Pseudorandom correlation generator: $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like n samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$ ’ 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.

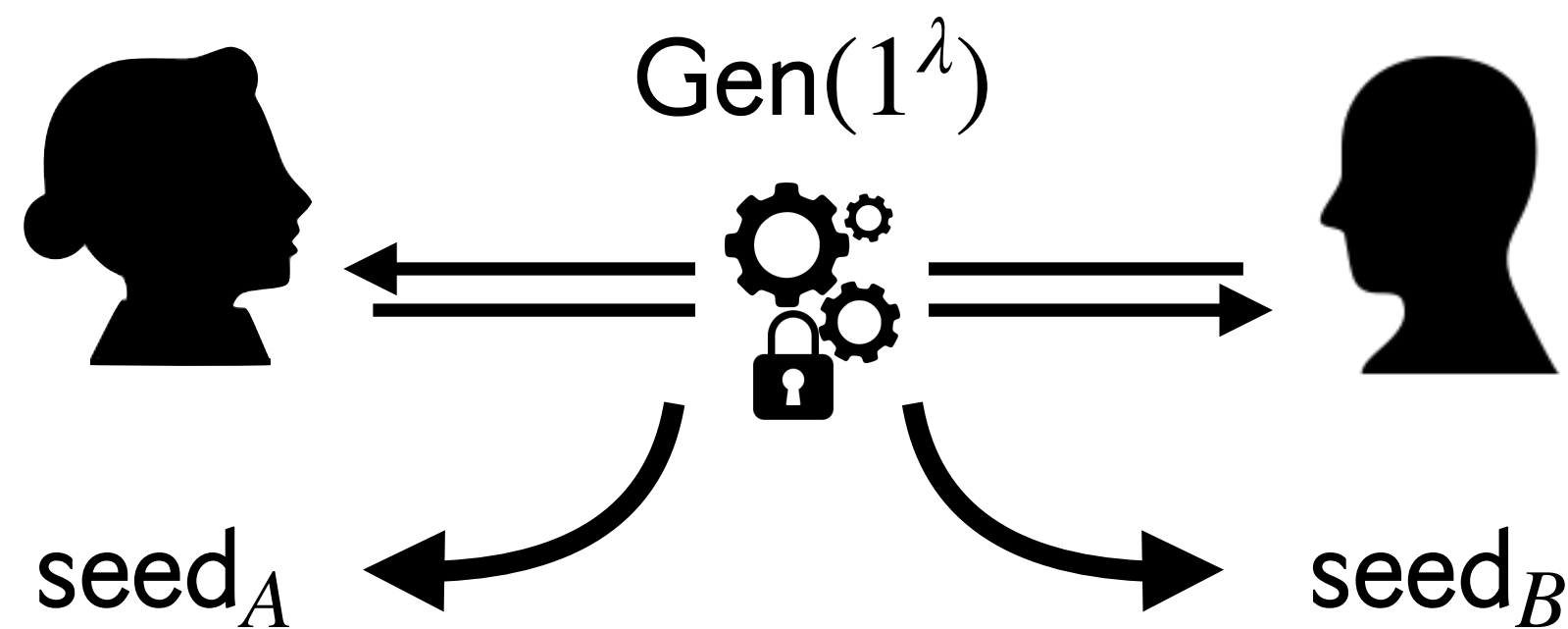
Preprocessing phase

Online phase

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

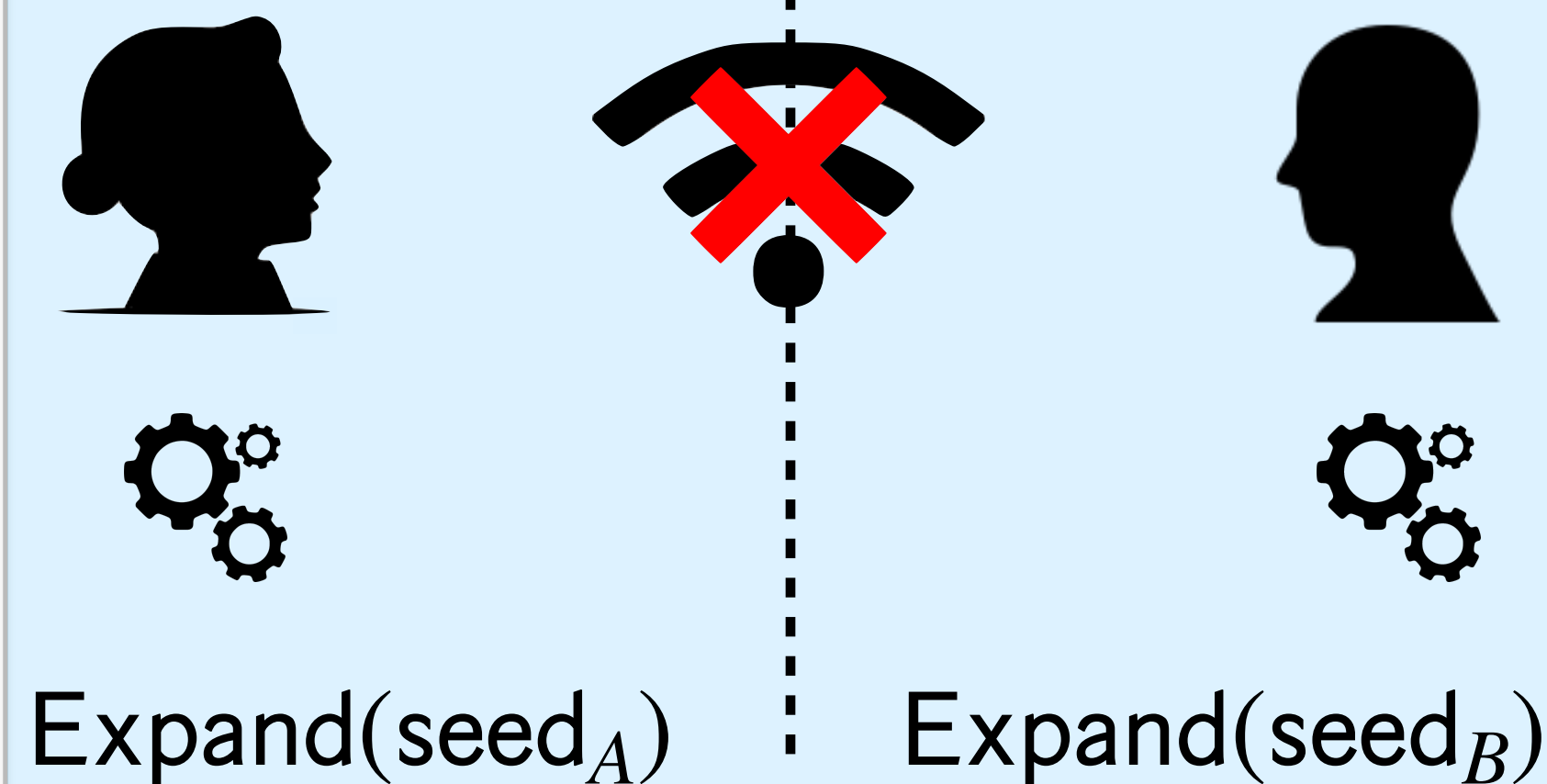
Pseudorandom correlation generator: $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like n samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$ ’ 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.

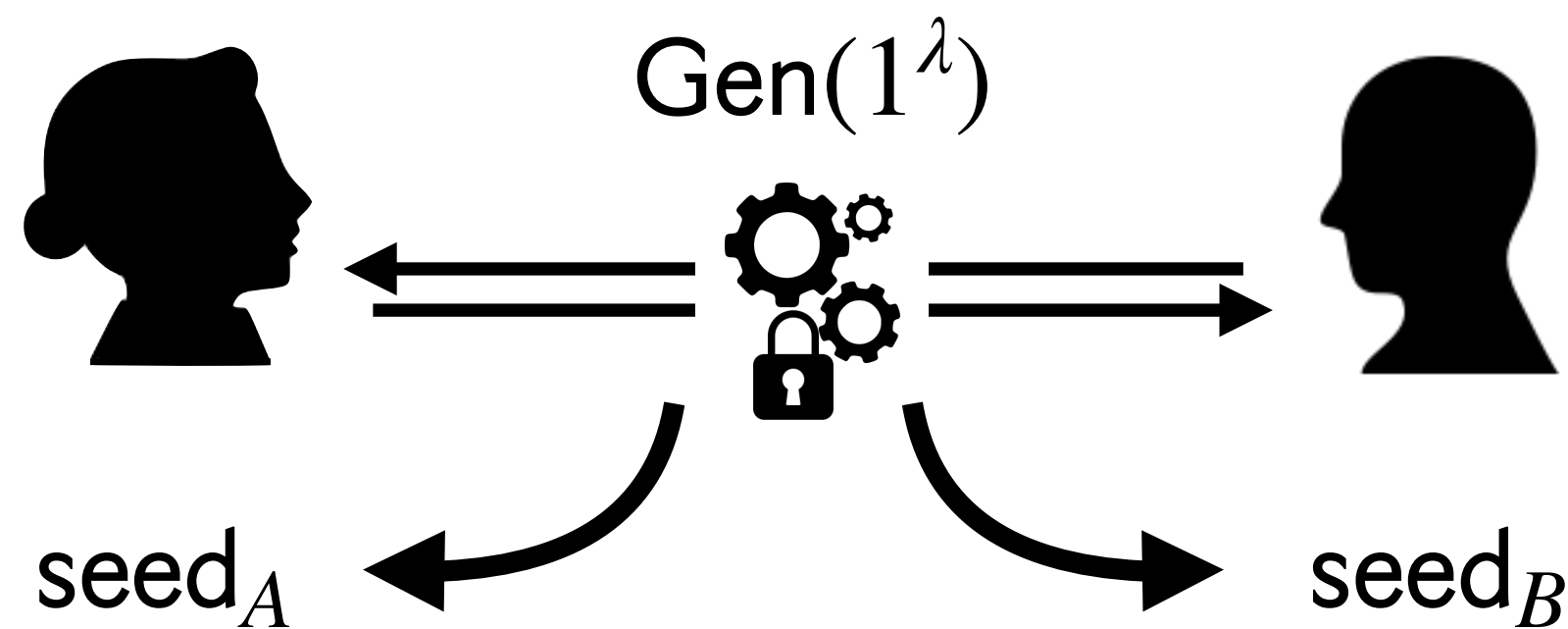
Preprocessing phase

Online phase

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

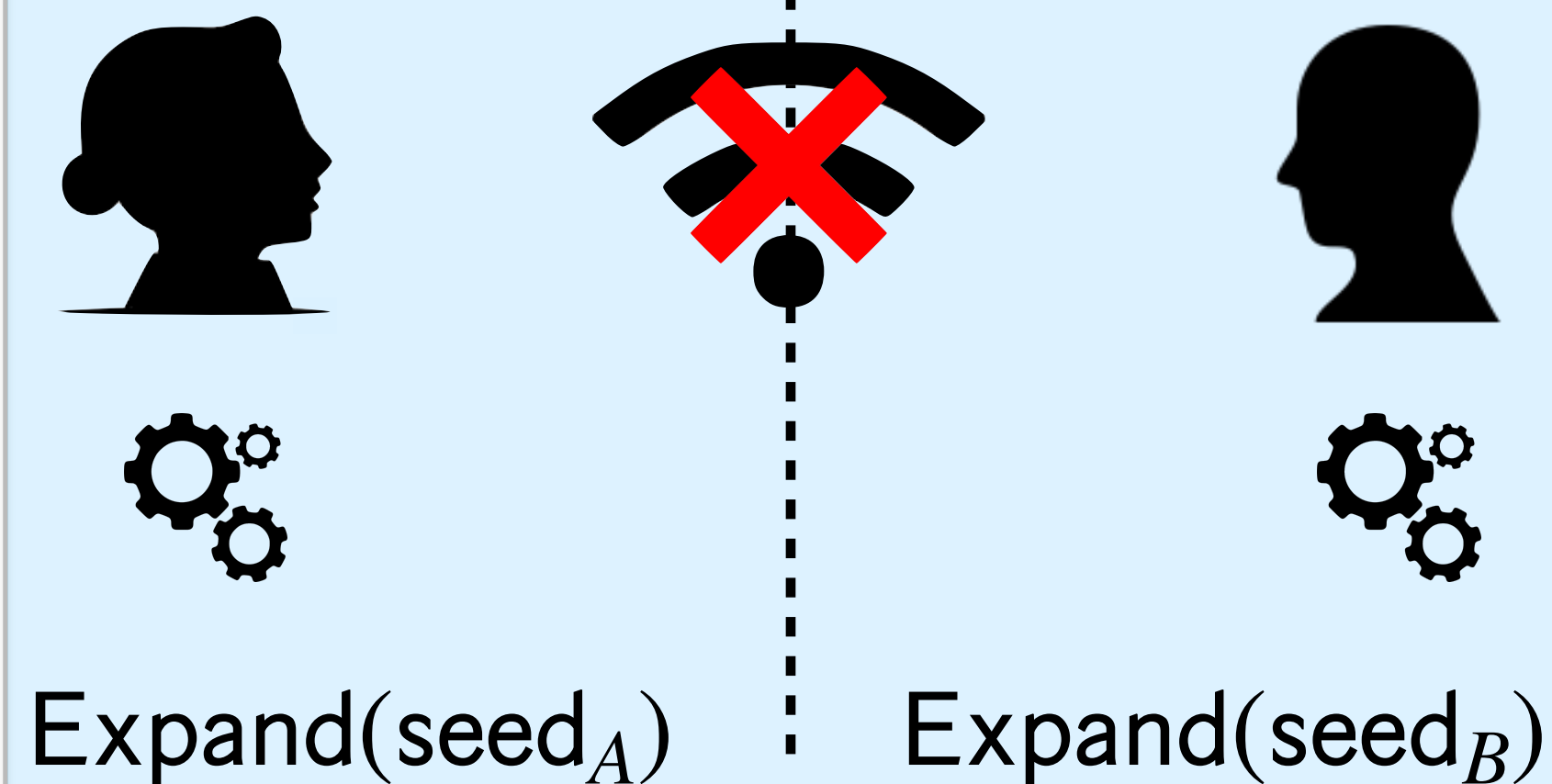
Pseudorandom correlation generator: $\text{Gen}(1^\lambda) \rightarrow (\text{seed}_A, \text{seed}_B)$ such that (1) $(\text{Expand}(A, \text{seed}_A), \text{Expand}(B, \text{seed}_B))$ looks like n samples from the target correlation, and (2) $\text{Expand}(A, \text{seed}_A)$ looks ‘random conditioned on satisfying the correlation with $\text{Expand}(B, \text{seed}_B)$ ’ 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.

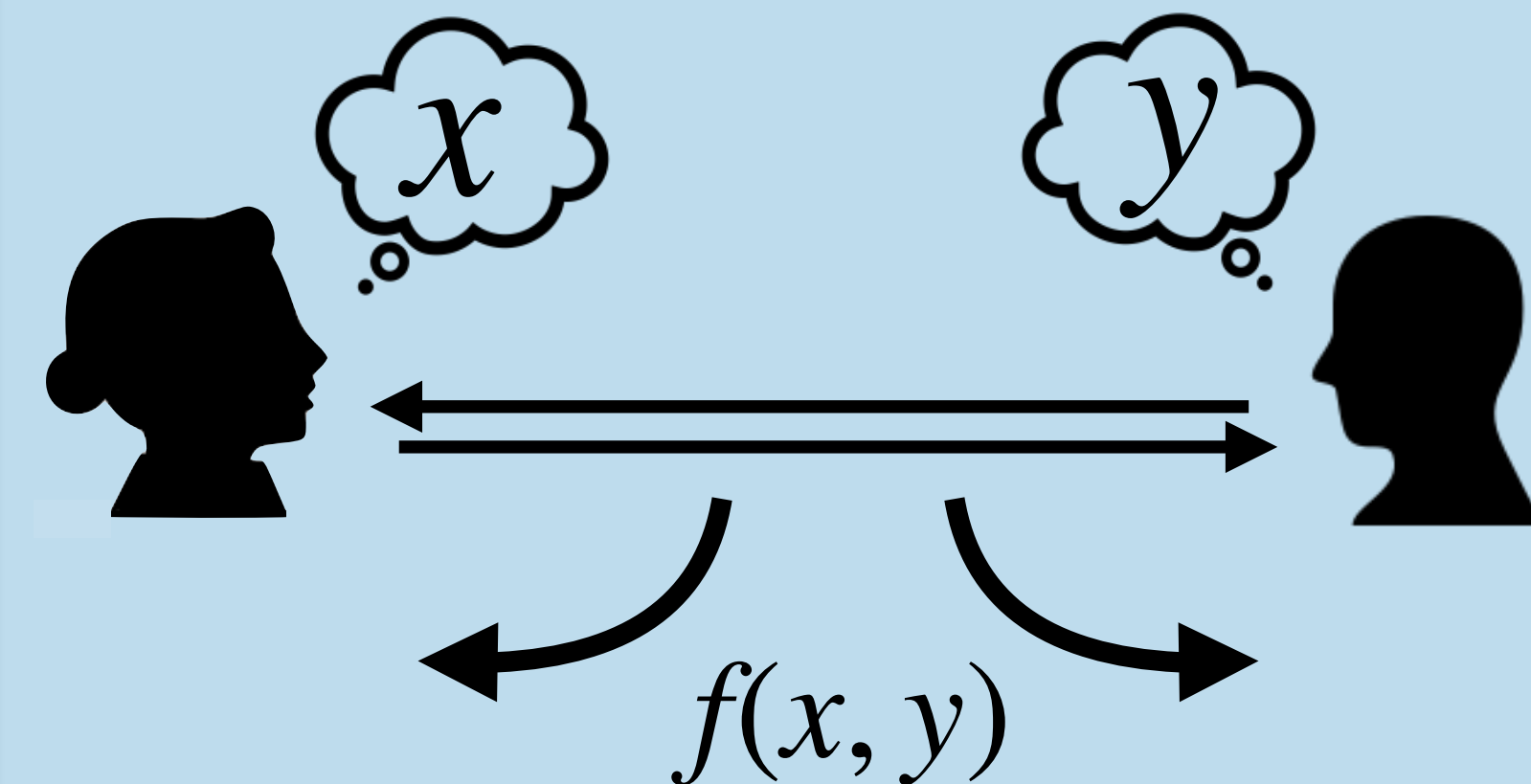
‘Silent’ computation



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?



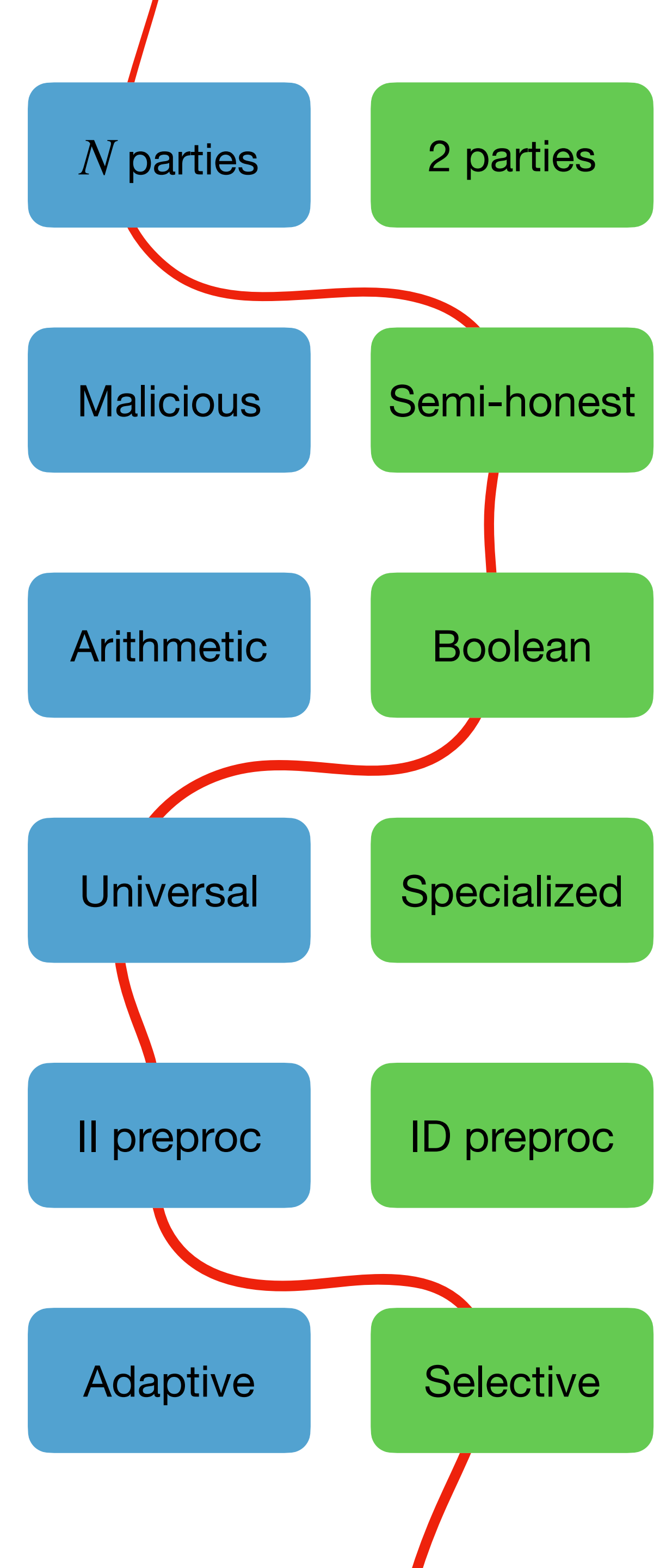
*One does not simply build
some MPC protocol*

Q: What Correlations C do we Consider?

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



*One does not simply build
some MPC protocol*



Q: What Correlations C do we Consider?

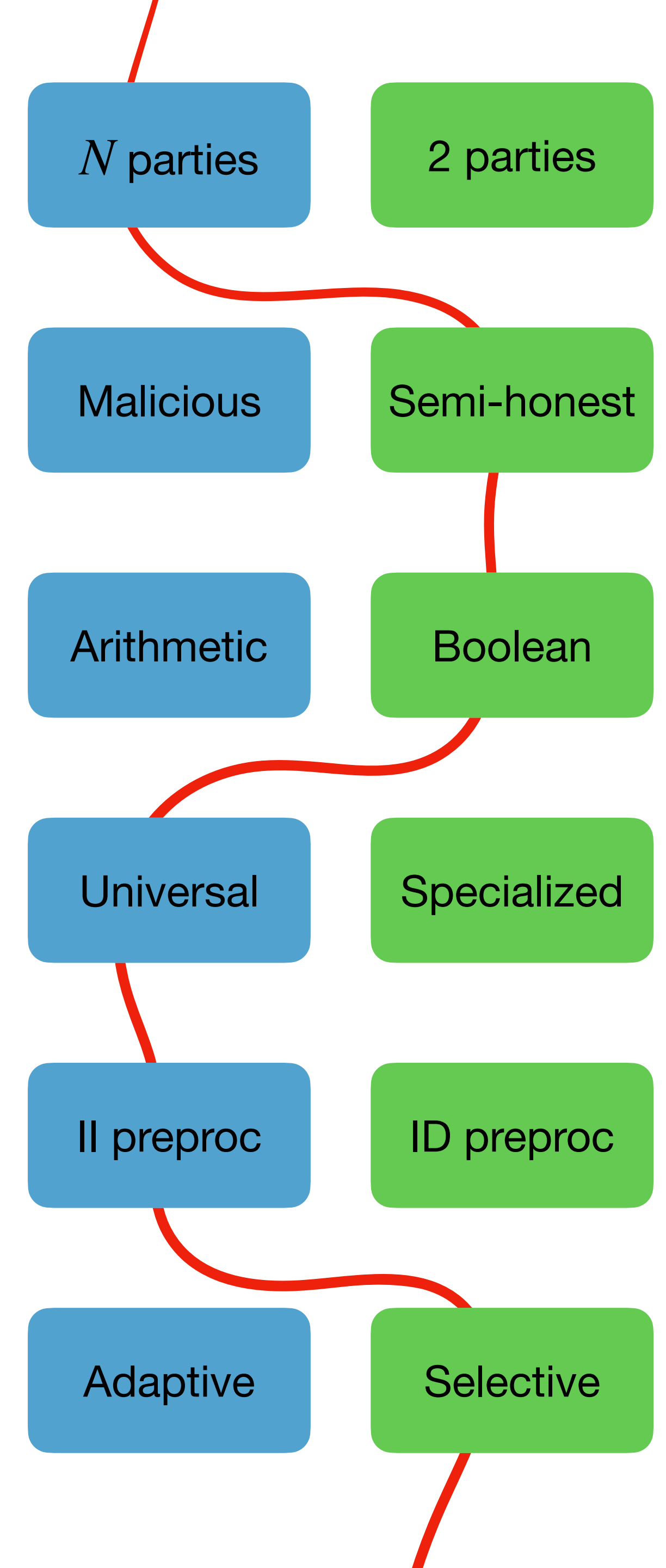
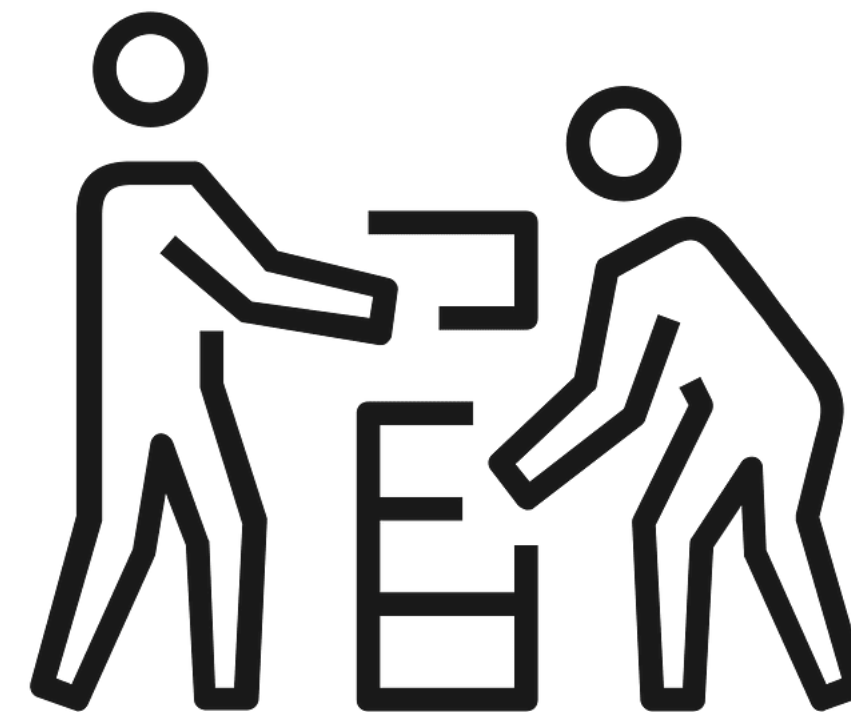
R: It depends! What MPC Protocol do you Want?



*One does not simply build
some MPC protocol*

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...



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

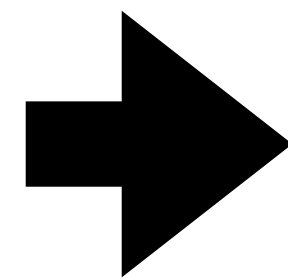
Given a correlation C , the dealer distributes shares of $C(r)$



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

Correlation (function)

Shares $\rightarrow \langle C(r) \rangle$
Random coin $\rightarrow$



Public function

$\underbrace{\hspace{1cm}}_{\text{Function secret sharing}} \underbrace{\hspace{1cm}}_{\text{Public function}} \underbrace{\hspace{1cm}}_{\text{Short seed}}$
 $FSS(C \circ PRG(\text{seed}))$

Function secret sharing

Short seed

A Template to Instantiate *Efficiently* the Correlated Randomness Model

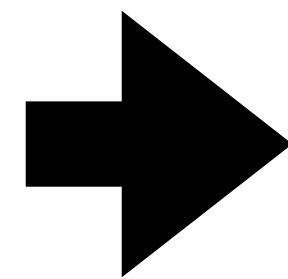
Given a correlation C , the dealer distributes shares of $C(r)$



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

Correlation (function)

Shares $\rightarrow \langle C(r) \rangle$
Random coin $\rightarrow$



Public function

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

Function secret sharing

Short seed

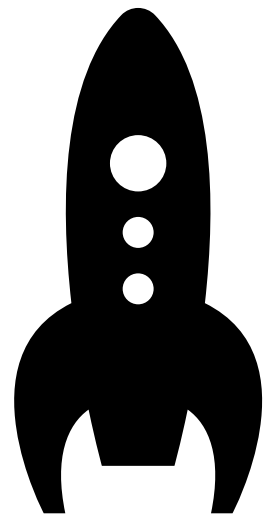
Function Secret Sharing

$$\text{FSS} \left\{ \begin{array}{l} \text{Share}(f) \mapsto (\text{ } \text{ } \text{ }) \\ \text{Eval}(\text{ } , x) + \text{Eval}(\text{ } , x) = f(x) \end{array} \right.$$

Function Secret Sharing

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

Low end



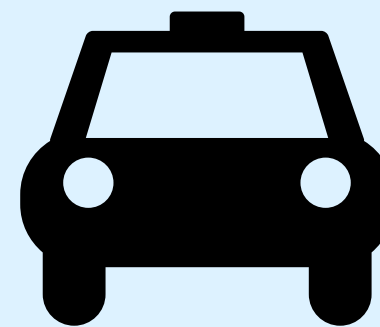
OWF

IT

Point functions

Lin. comb.

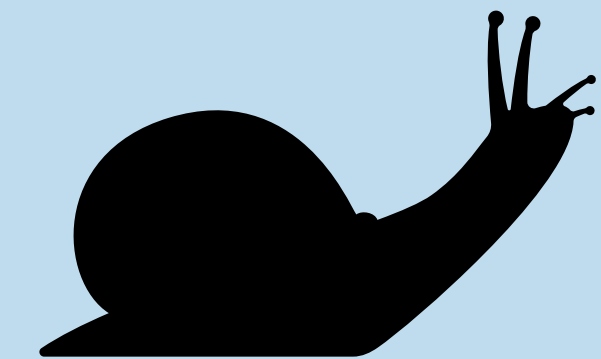
Mid end



DDH, DCR, class groups

NC^1

High end



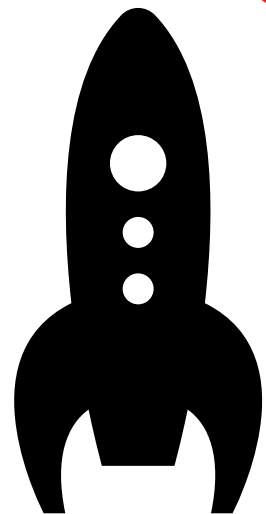
iO, Multi-key threshold FHE

All polytime functions

Function Secret Sharing

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

Low end



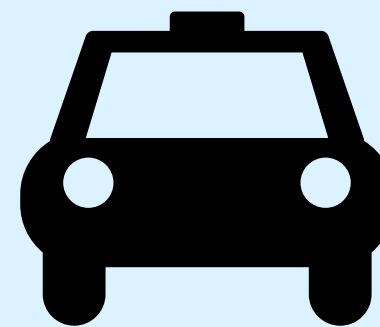
OWF

IT

Point functions

Lin. comb.

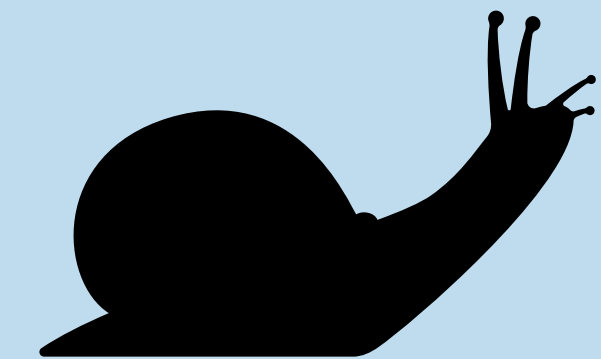
Mid end



DDH, DCR, class groups

NC^1

High end

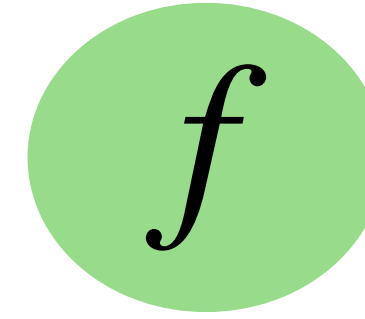


iO, Multi-key threshold FHE

All polytime functions

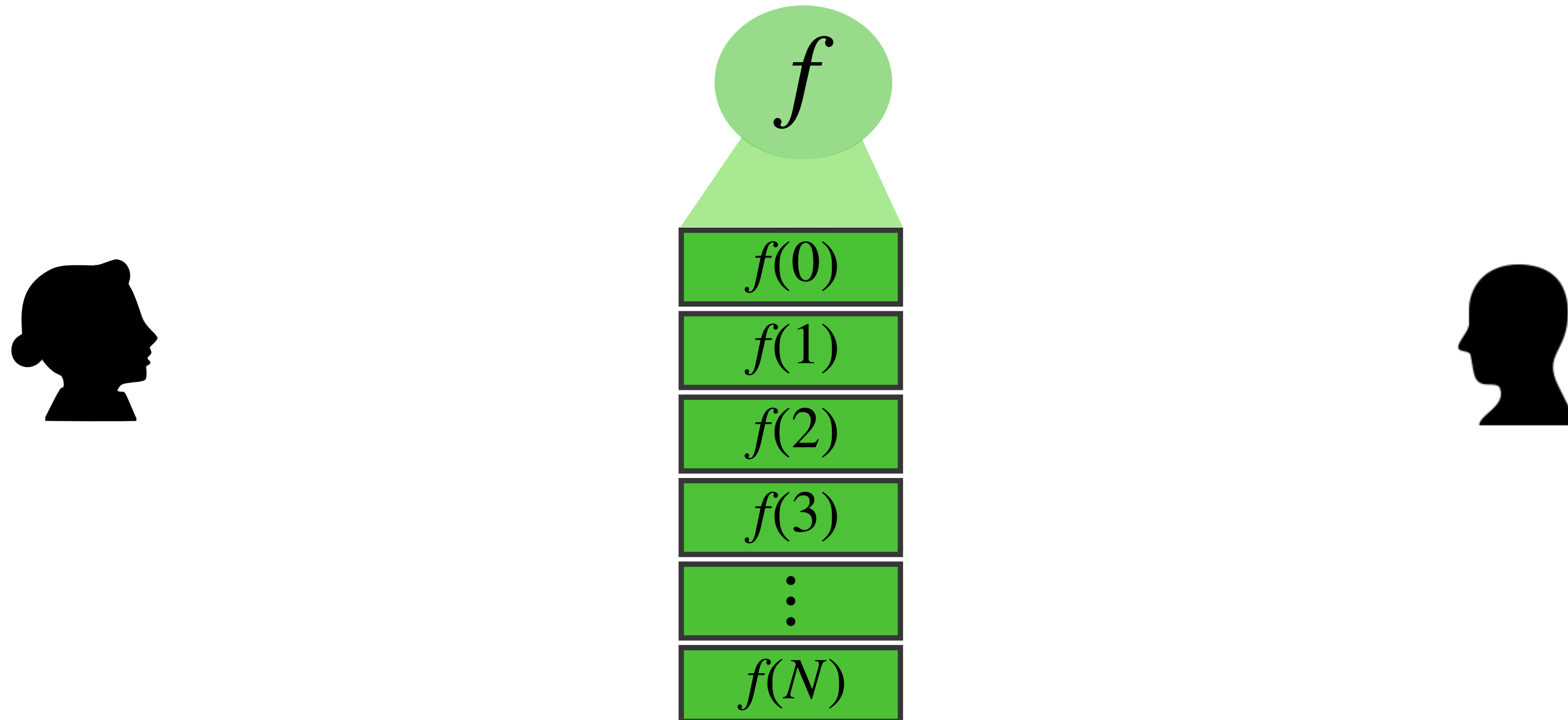
What Functions can be Shared?

Sharing an arbitrary function:



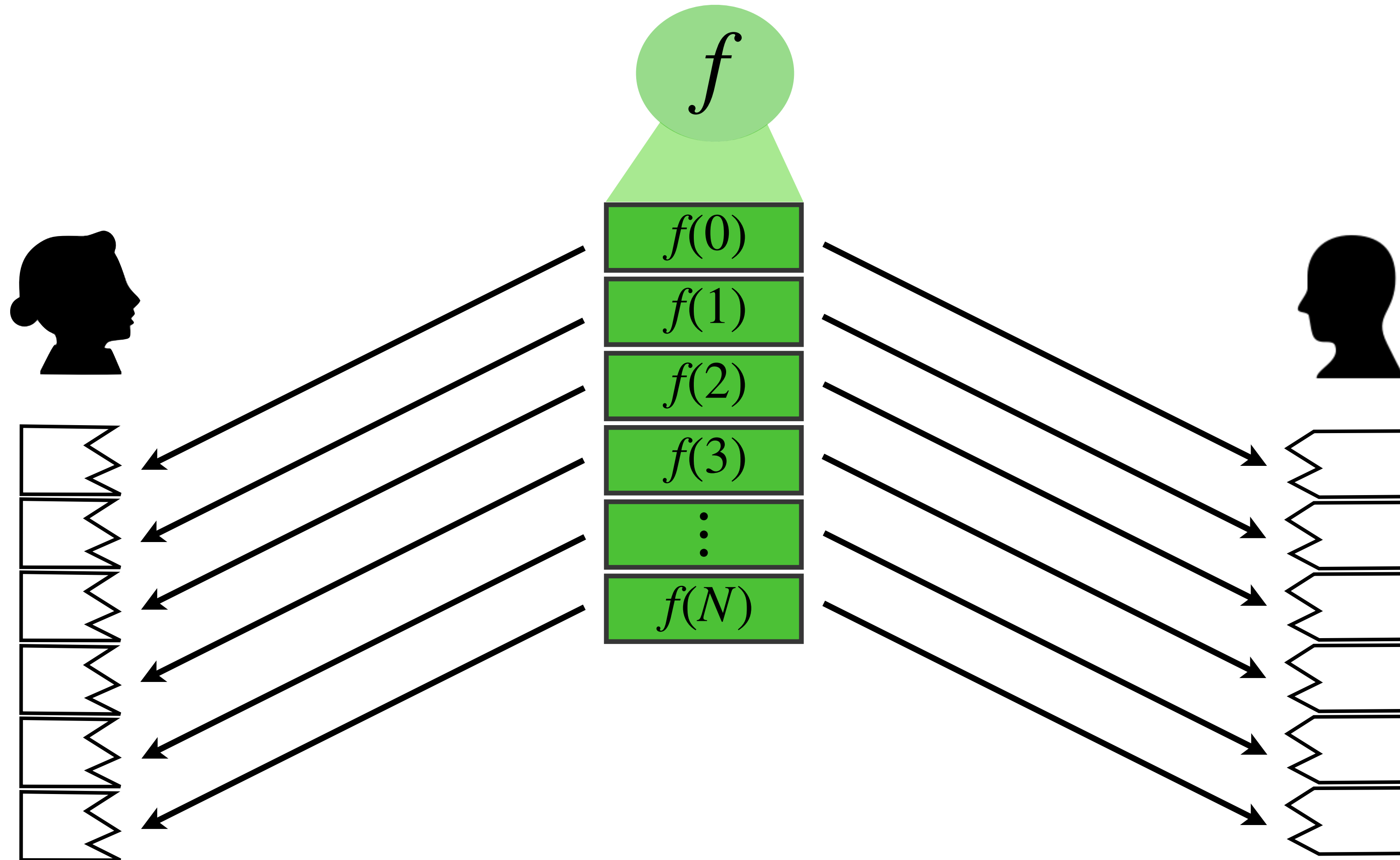
What Functions can be Shared?

Sharing an arbitrary function:



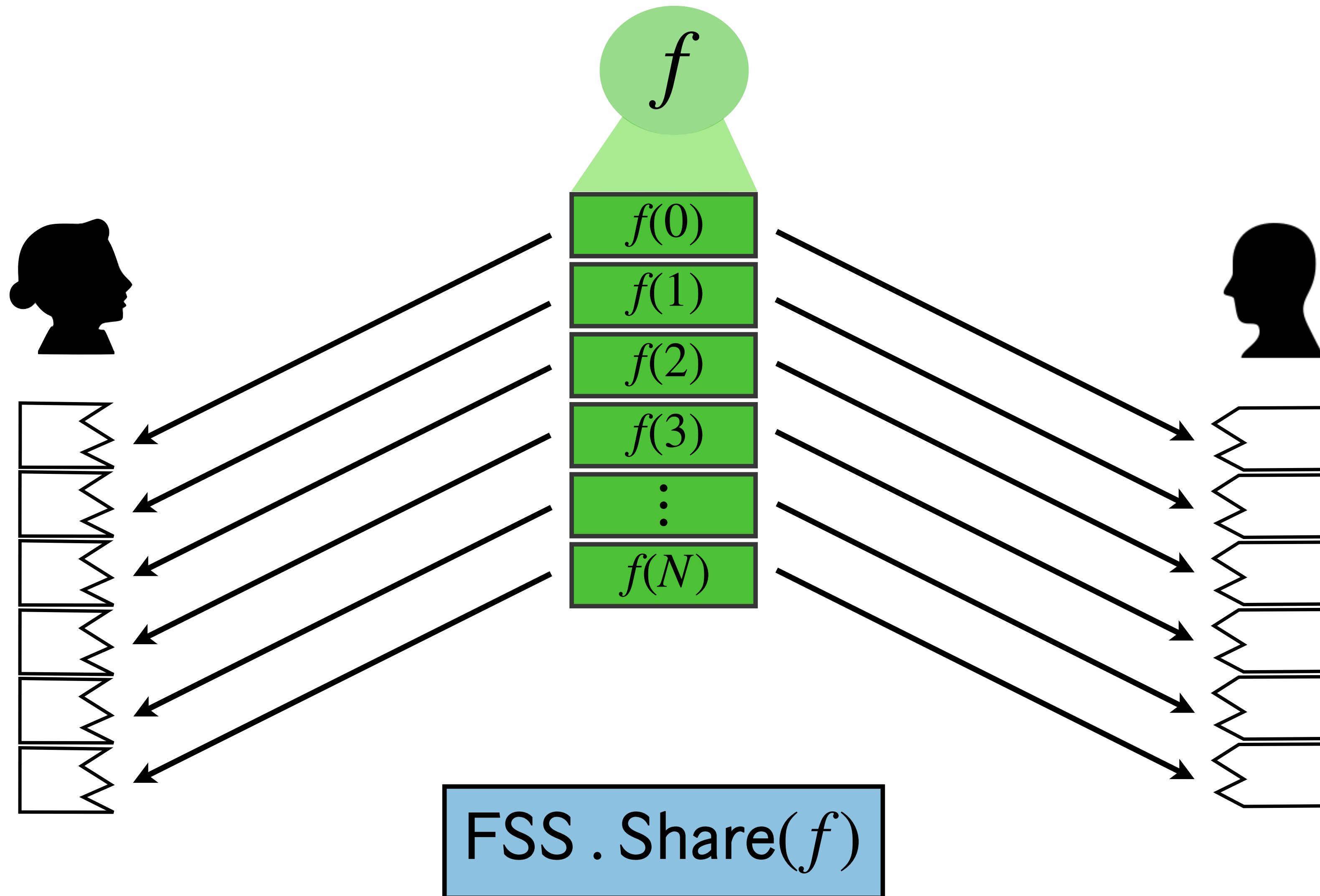
What Functions can be Shared?

Sharing an arbitrary function:



What Functions can be Shared?

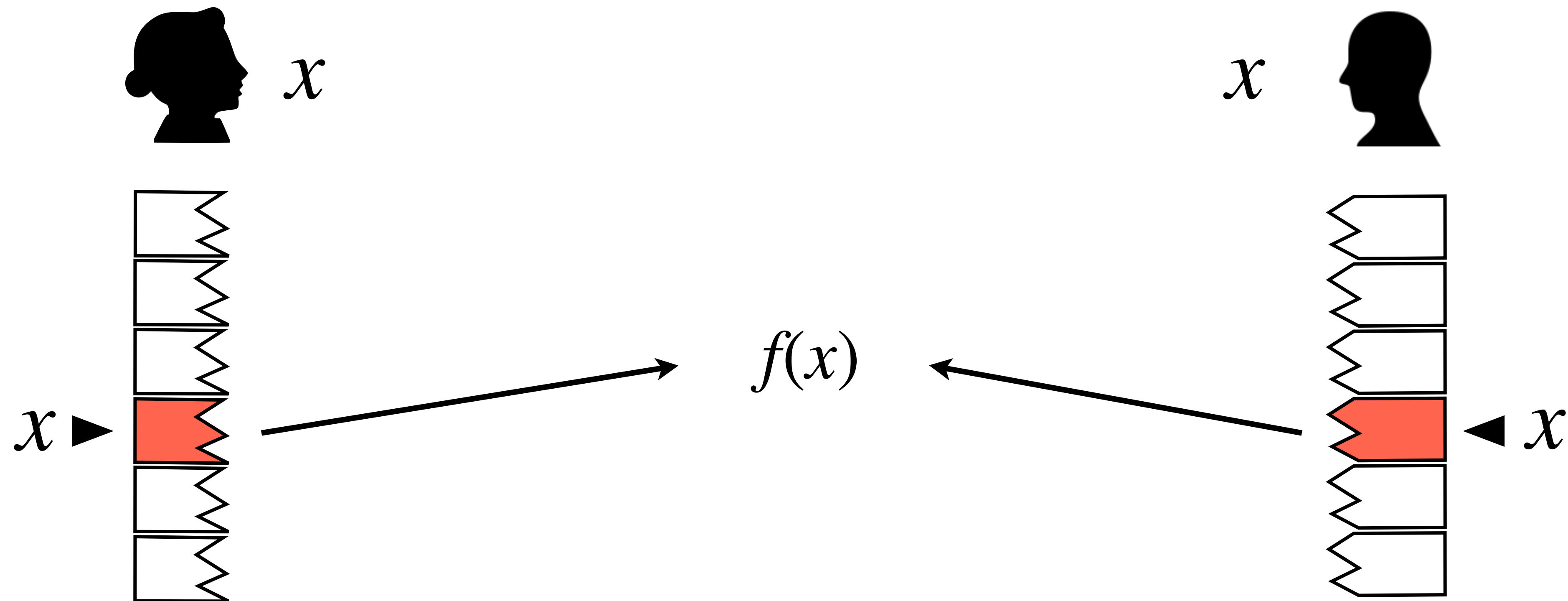
Sharing an arbitrary function:



What Functions can be Shared?

Sharing an arbitrary function:

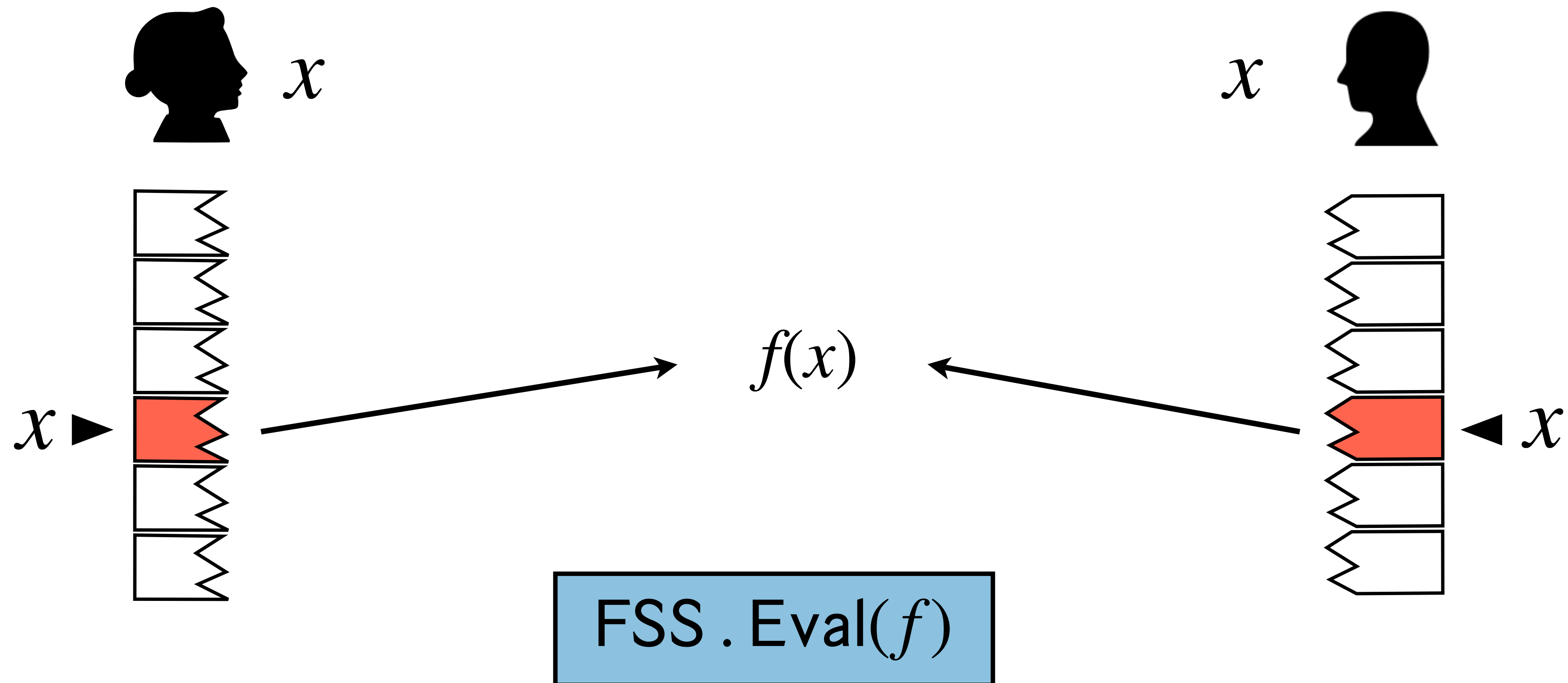
f



What Functions can be Shared?

Sharing an arbitrary function:

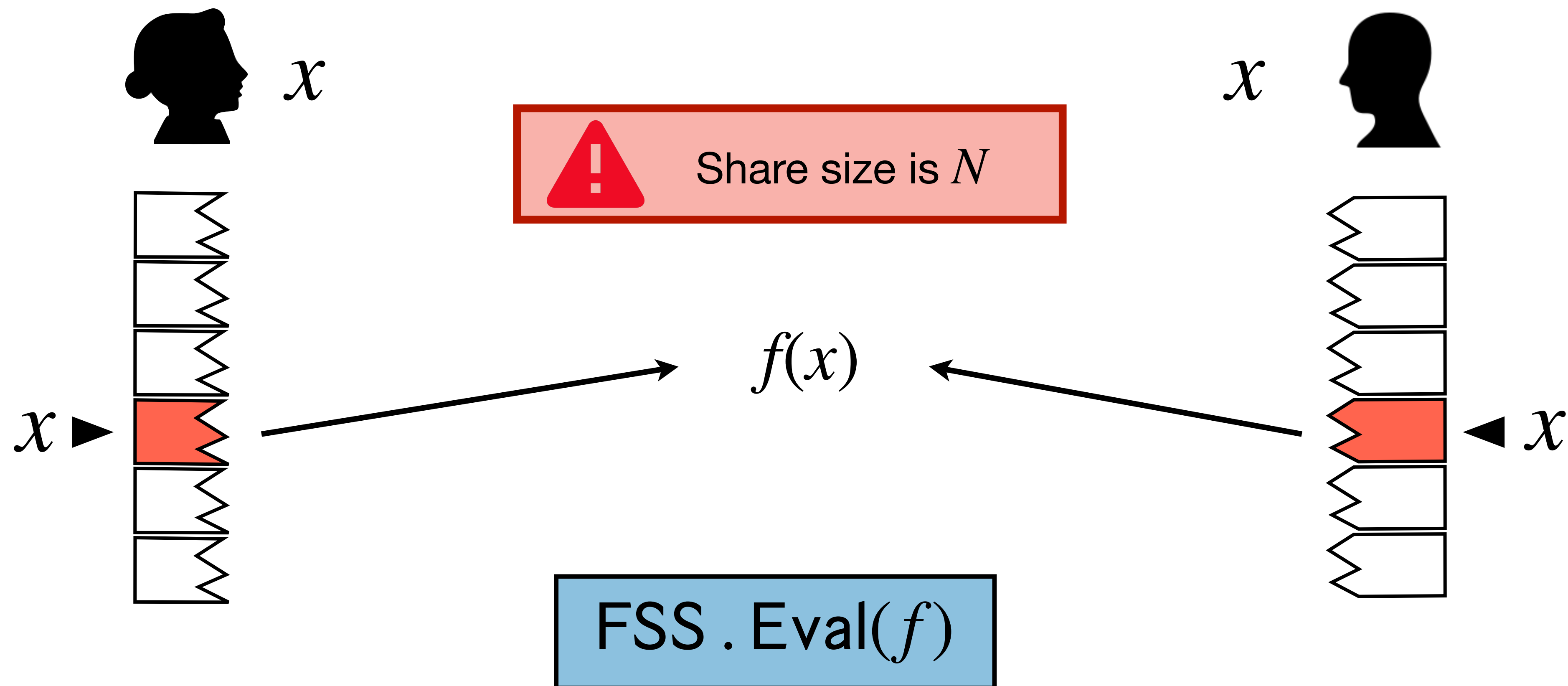
f



What Functions can be Shared?

Sharing an arbitrary function:

f



What Functions can be Shared?

succinctly



What Functions can be Shared?

succinctly

Sharing the all zero function:

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

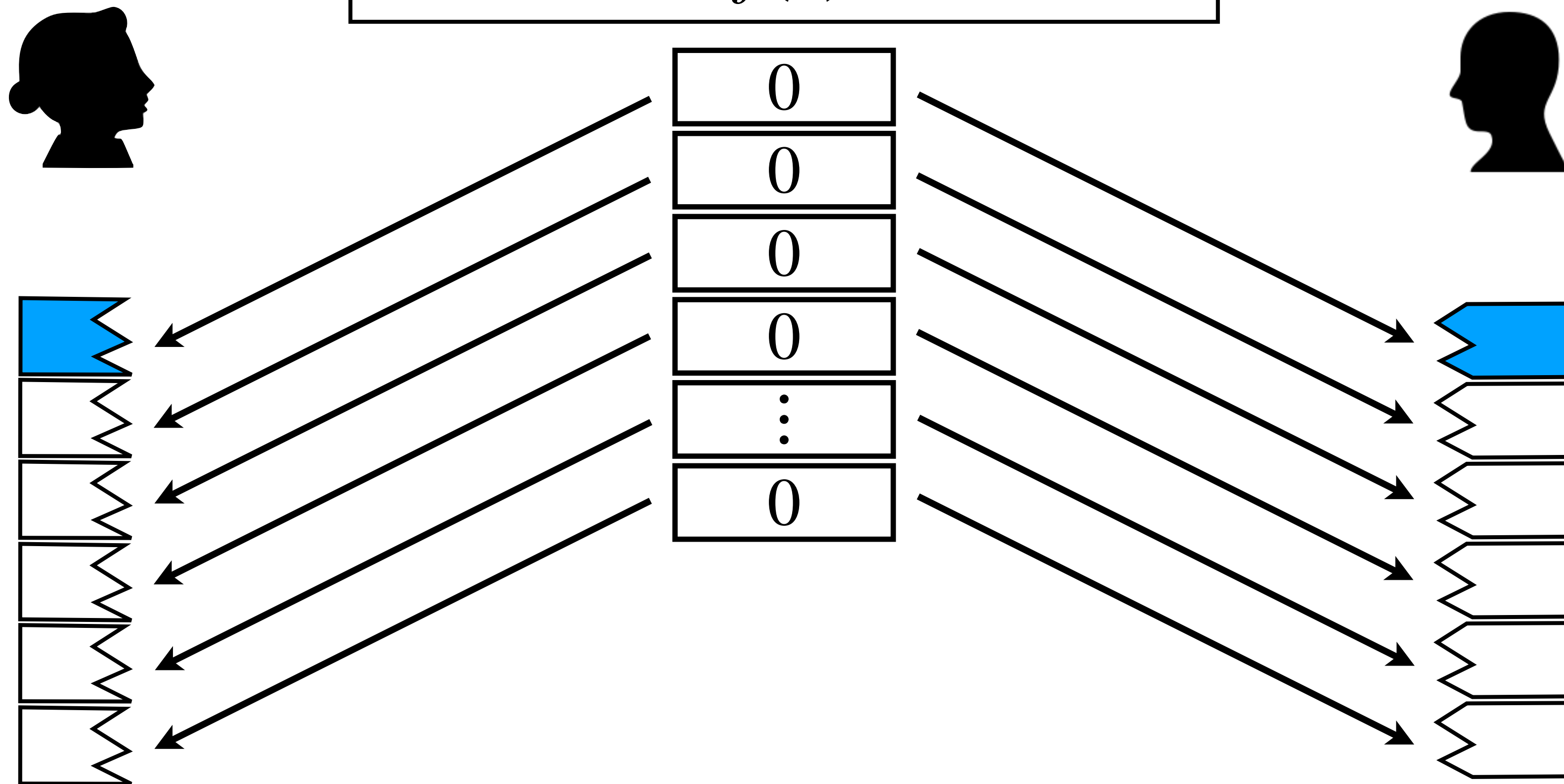


What Functions can be Shared?

succinctly

Sharing the all zero function:

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

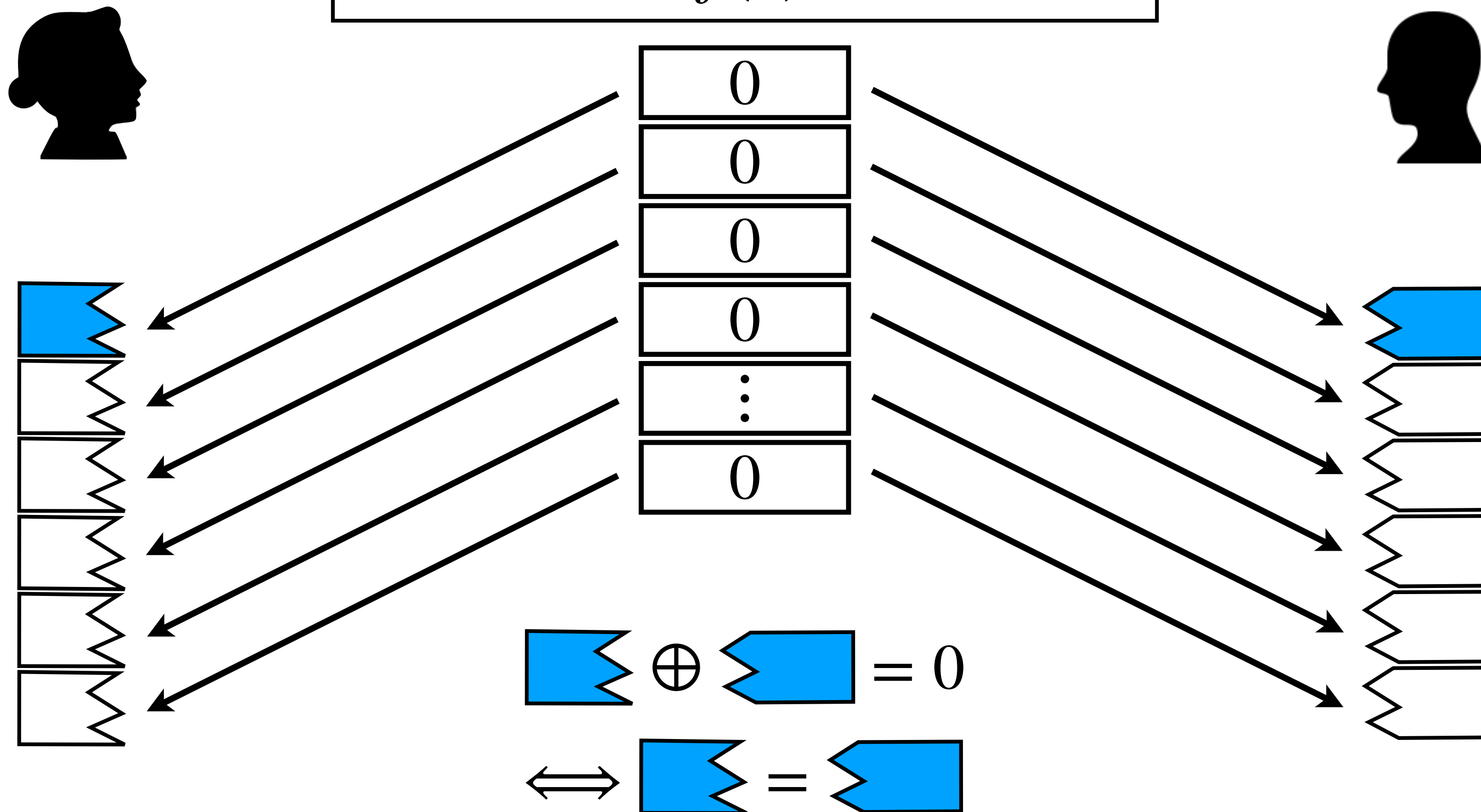


What Functions can be Shared?

succinctly

Sharing the all zero function:

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



What Functions can be Shared?

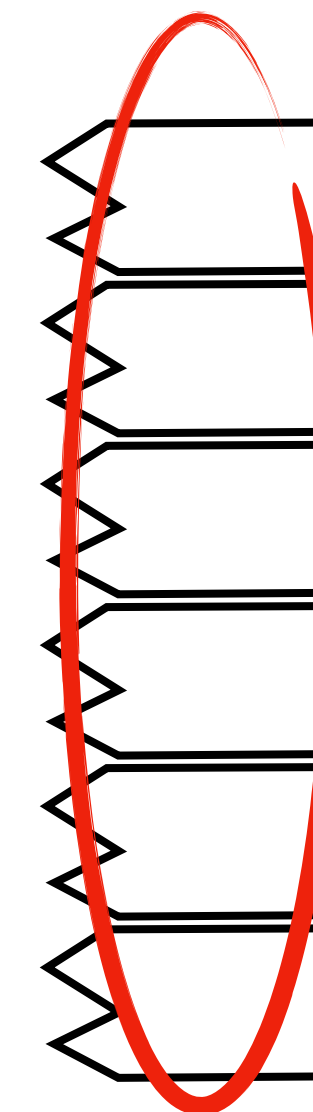
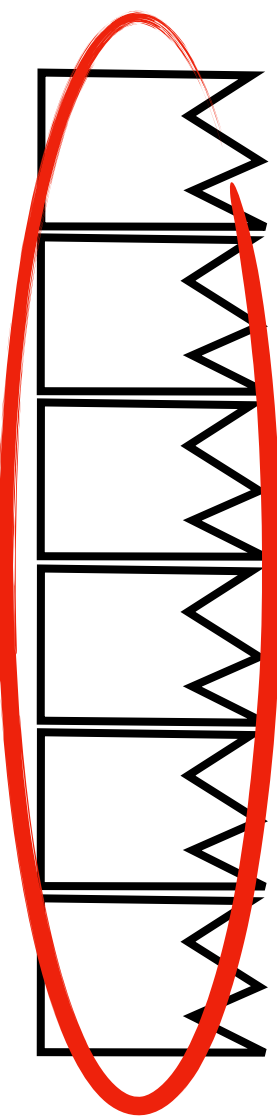
succinctly

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:

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



What Functions can be Shared?

succinctly

Sharing a *point function*

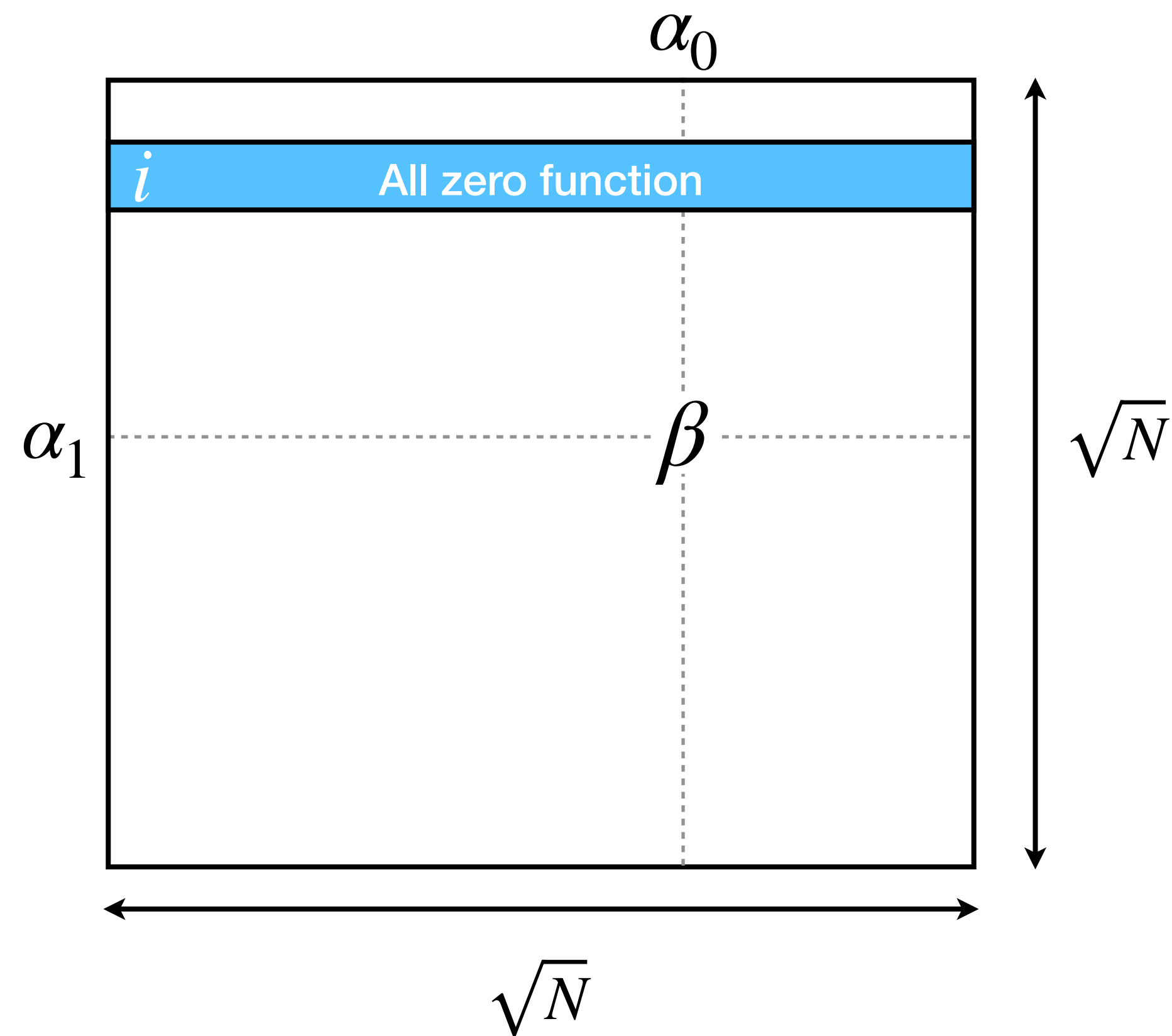
$$f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$$



What Functions can be Shared?

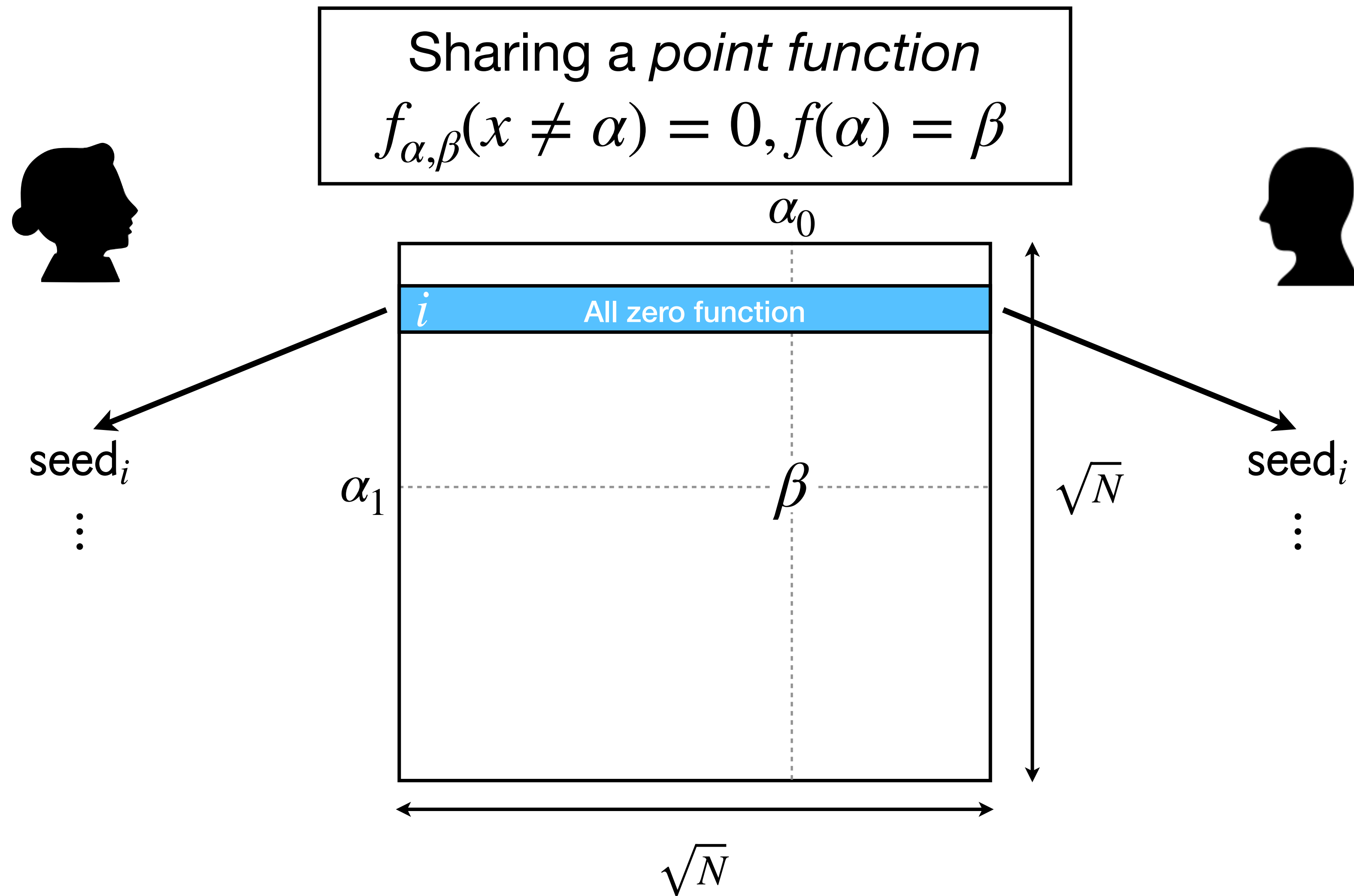
succinctly

Sharing a *point function*
 $f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$



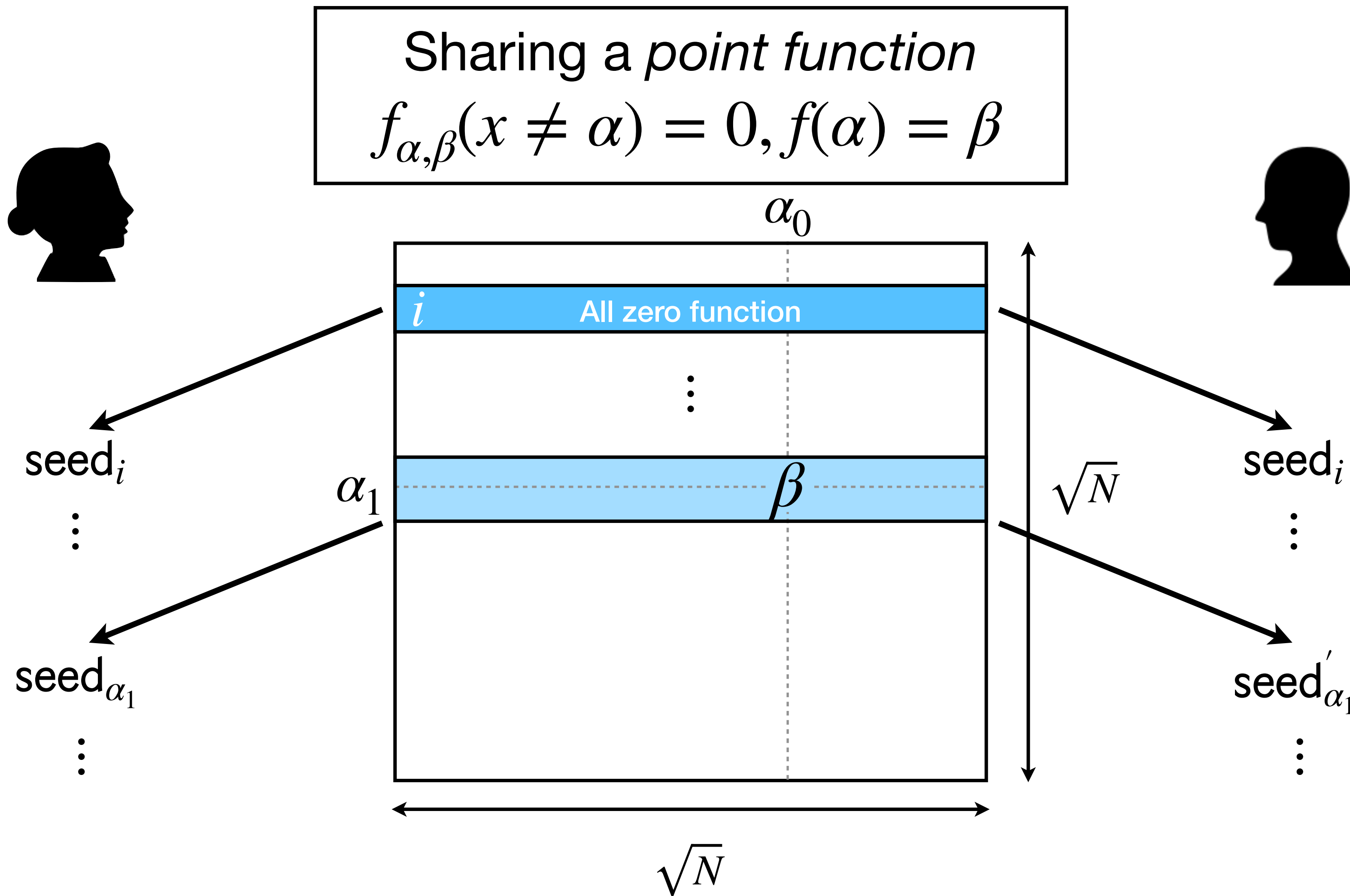
What Functions can be Shared?

succinctly



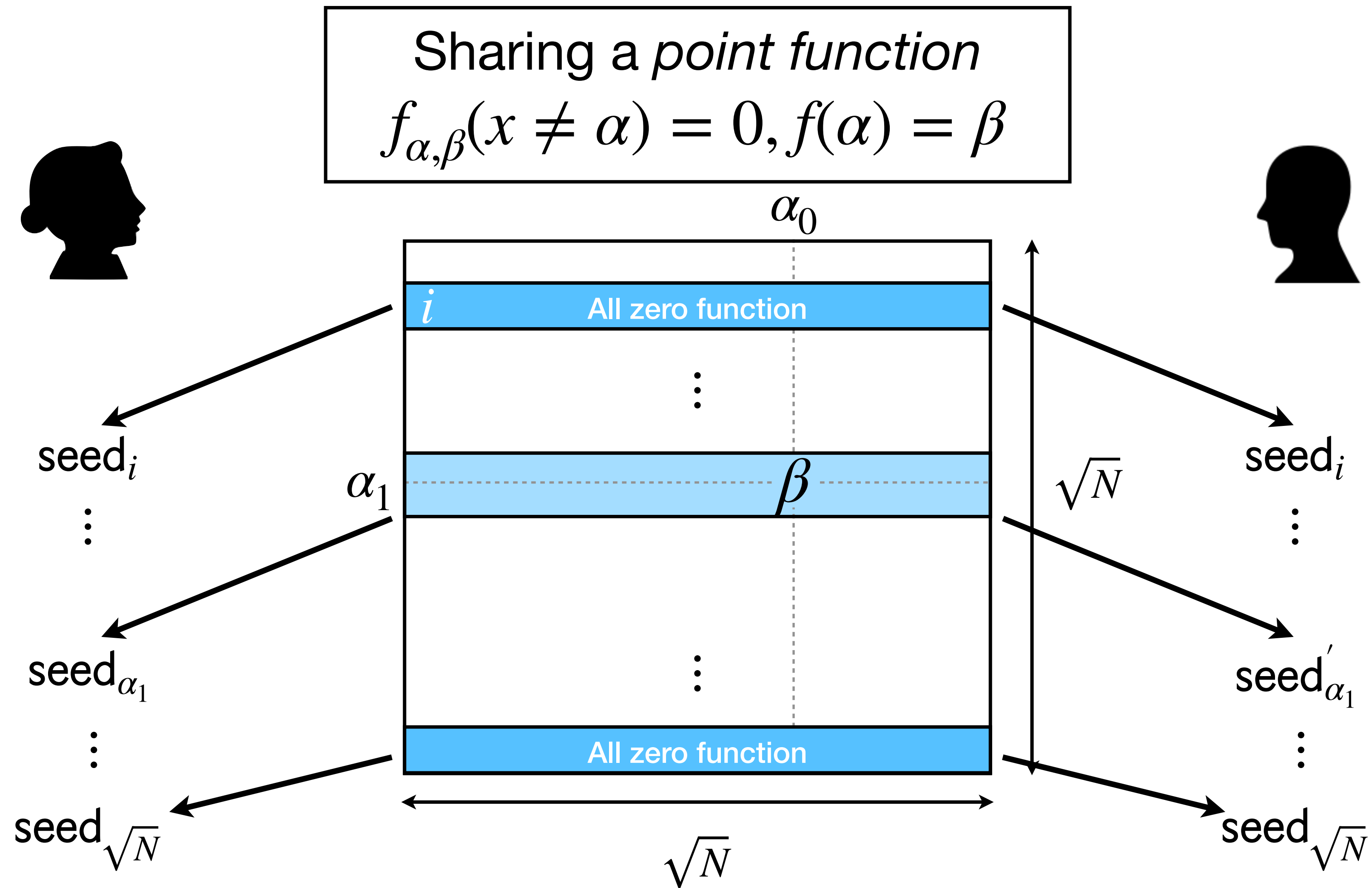
What Functions can be Shared?

succinctly



What Functions can be Shared?

succinctly



What Functions can be Shared?

succinctly

Sharing a *point function*
 $f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$



Public

seed_{*i*}

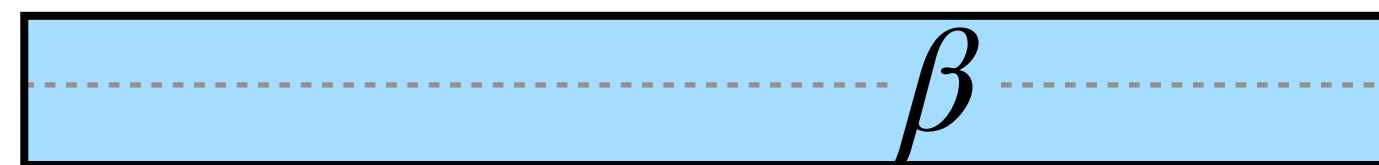
⋮

seed _{α_1}

⋮

seed _{$\sqrt{N}$}

$\Delta =$



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

seed_{*i*}

⋮

seed' _{α_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

b_i seed _{i}
⋮
 b_{α_1} seed _{α_1}
⋮
 $b_{\sqrt{N}}$ seed _{$\sqrt{N}$}

$$\Delta =$$

β

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

seed _{i} b_i
⋮
seed _{α_1} ' $1 - b_{\alpha_1}$
⋮
seed _{$\sqrt{N}$} $b_{\sqrt{N}}$

What Functions can be Shared?

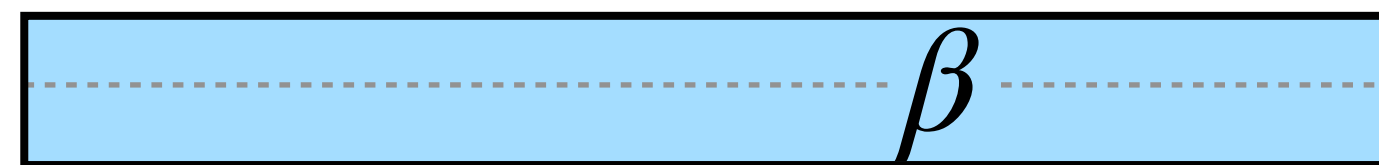
succinctly

Sharing a *point function*
 $f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$



Public

$\Delta =$



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

b_i seed_{*i*}
⋮

seed_{*i*} b_i
⋮

b_{α_1} seed _{α_1}
⋮

seed' _{α_1} $1 - b_{\alpha_1}$
⋮

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

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

Eval :

$$\text{PRG}(\text{seed}_j) \oplus \Delta \cdot b_j$$

What Functions can be Shared?

succinctly

Sharing a *point function*
 $f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$



Recurring:

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

b_i seed _{i}
⋮

seed _{i} b_i
⋮

b_{α_1} seed _{α_1}
⋮

seed _{α_1} $1 - b_{\alpha_1}$
⋮

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

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

What Functions can be Shared?

succinctly

Sharing a *point function*
 $f_{\alpha,\beta}(x \neq \alpha) = 0, f(\alpha) = \beta$



Recurring:

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

b_i seed _{i}
 ⋮

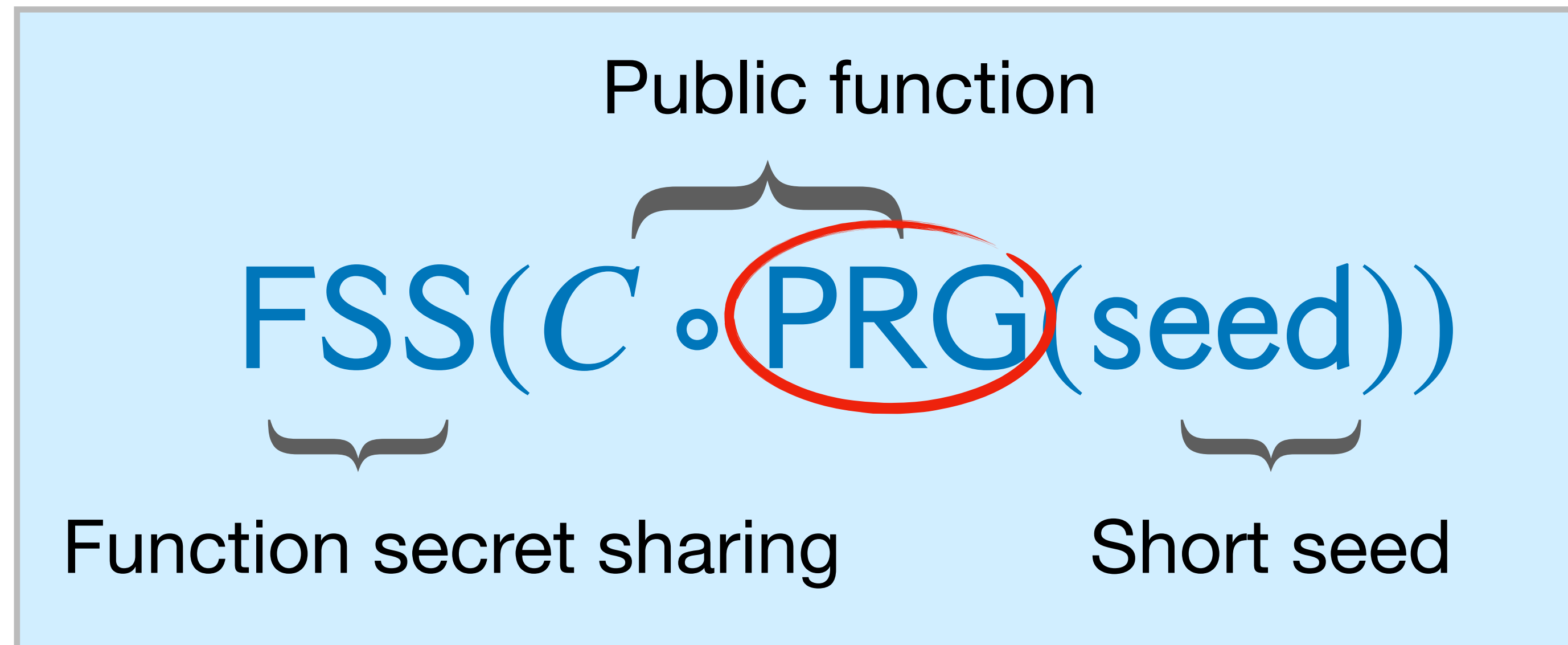
seed _{i} b_i
 ⋮

b_{α_1} seed _{α_1}
 ⋮
 $b_{\sqrt{N}}$ seed _{$\sqrt{N}$}

seed _{α_1} $1 - b_{\alpha_1}$
 ⋮
seed _{$\sqrt{N}$} $b_{\sqrt{N}}$

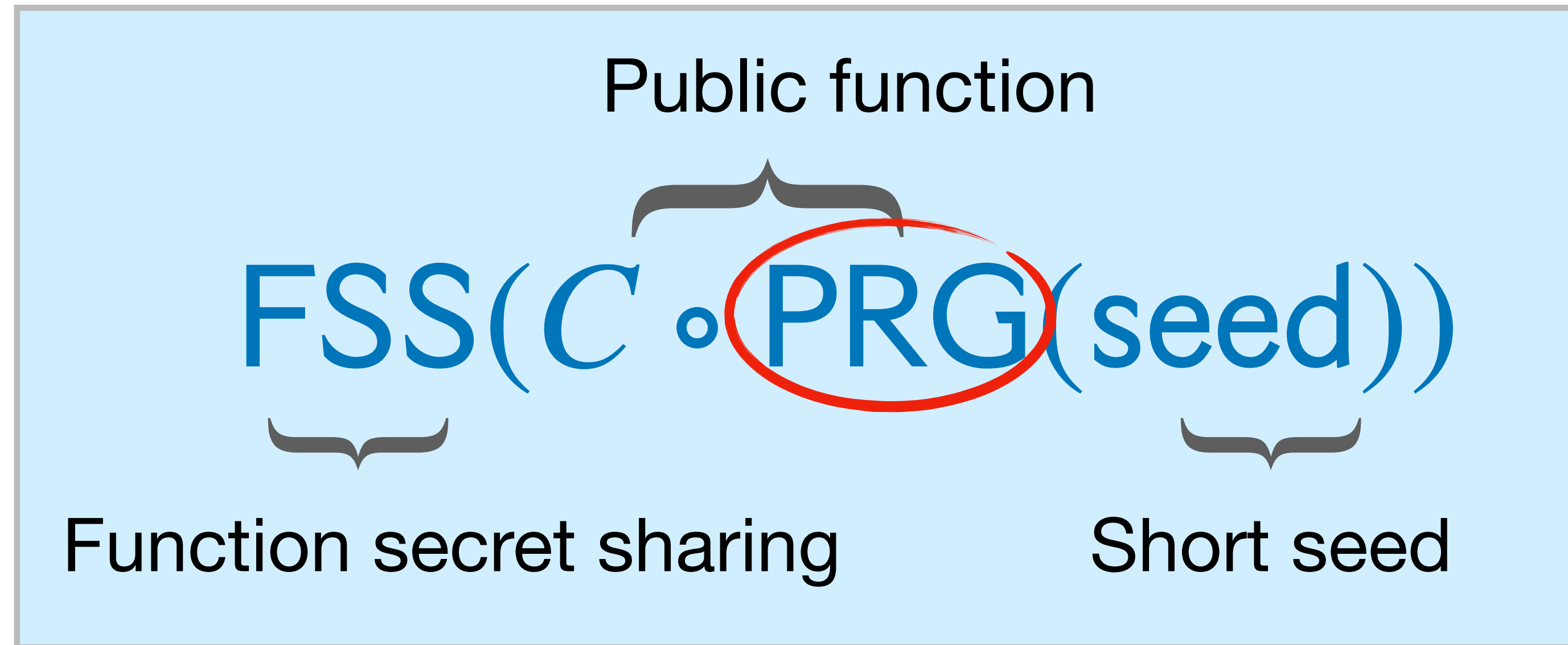
This + later improvements [BGI16]:
FSS for point functions with keys of size
 $O(\lambda \cdot \log N)$

Back to the PCG Template



We can succinctly share point functions

Back to the PCG Template

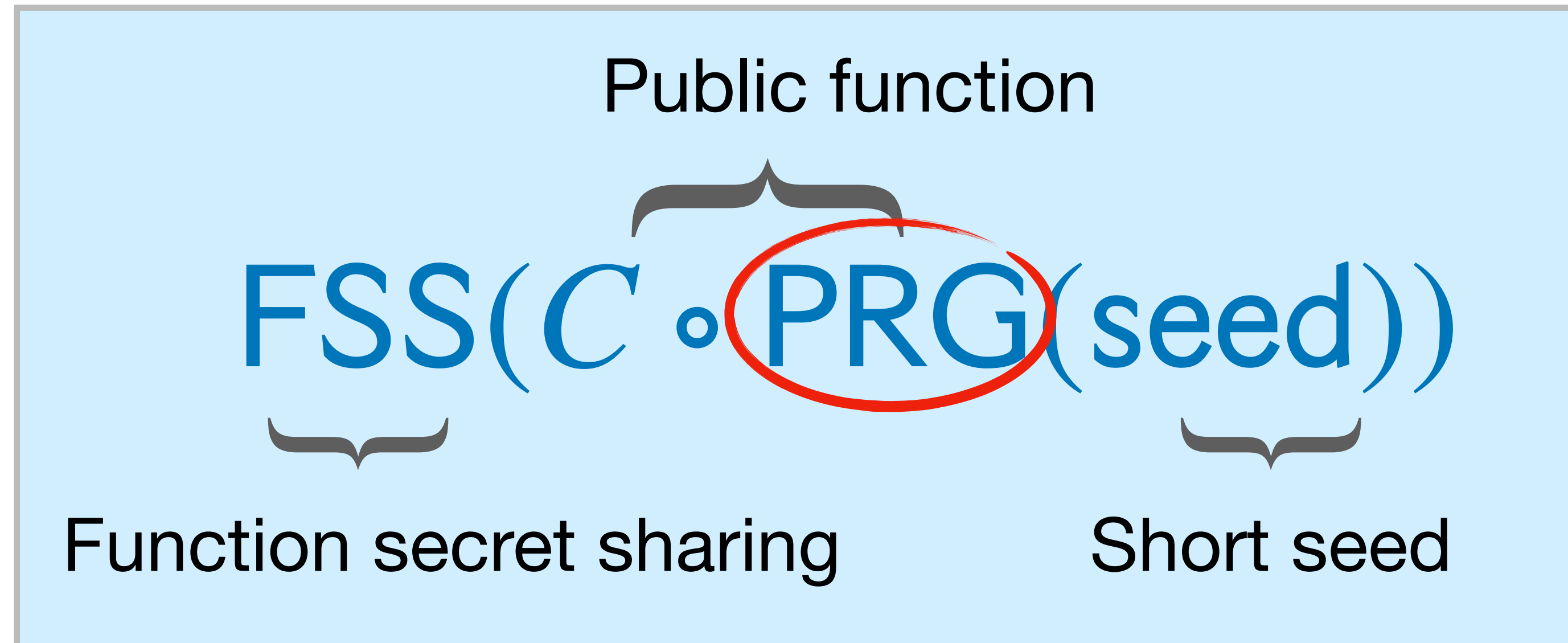


Secret sharing
is additively
homomorphic!

We can succinctly share point functions

Linear combinations of

Back to the PCG Template



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

$$\left(\begin{array}{c} \text{Random matrix} \\ \uparrow \\ G \end{array} , \begin{array}{c} \text{Short secret} \\ \uparrow \\ G \cdot \text{Short secret} \end{array} + \begin{array}{c} \text{Sparse noise} \\ \uparrow \\ \text{Sparse noise} \end{array} \right) \approx \$$$

Random matrix Short secret Sparse noise

LPN to the Rescue

The LPN assumption - primal

The diagram illustrates the primal LPN assumption as a matrix equation. It features a blue square labeled H on the left, followed by a dot and a large left parenthesis. Inside the parenthesis are a green square labeled G and a comma. To the right of the comma is another green square labeled G , followed by a dot and a small gray vertical rectangle. This is followed by a plus sign and a pink vertical rectangle. A large right parenthesis closes the group, followed by an approximation symbol $\approx$ and a dollar sign $\$$. Below the diagram, four labels with arrows point to their respective components: 'Parity-check matrix of G ' points to H ; 'Random matrix' points to the first G ; 'Short secret' points to the gray rectangle; and 'Sparse noise' points to the pink rectangle.

$$H \cdot \left(G, G \cdot \text{Short secret} + \text{Sparse noise} \right) \approx \$$$

Parity-check matrix of G Random matrix Short secret Sparse noise

LPN to the Rescue

The LPN assumption - primal

The diagram illustrates the primal LPN assumption equation:

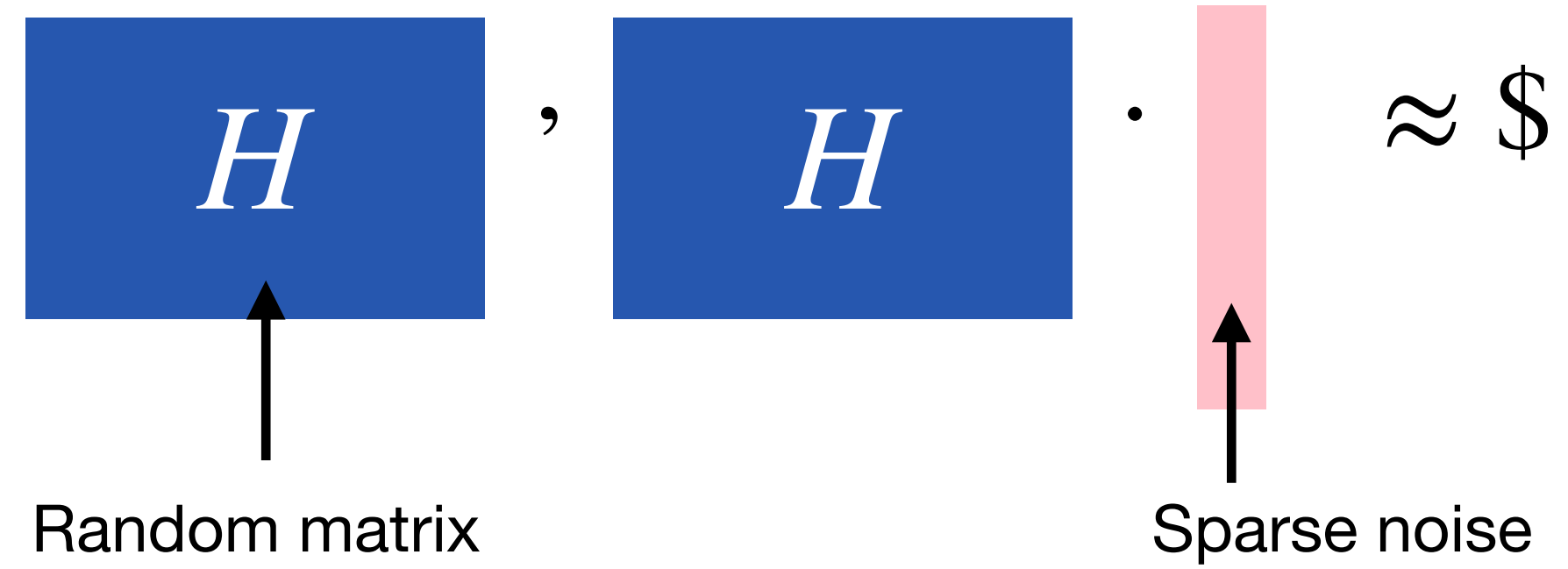
$$H \cdot \left(\text{Random matrix}, \text{Short secret} \cdot \text{Sparse noise} \right) \approx s$$

Labels below the diagram:

- Parity-check matrix of G (points to H)
- Random matrix (points to the first G in the parentheses)
- Short secret (points to s)
- Sparse noise (points to e)

LPN to the Rescue

The LPN assumption - dual



The diagram illustrates the LPN assumption - dual. It features two blue squares, each containing a white italicized H . The first square is labeled "Random matrix" with an upward-pointing arrow. A comma follows the first square. The second square is followed by a dot and a vertical pink rectangle. An upward-pointing arrow is labeled "Sparse noise" below the pink rectangle. To the right of the pink rectangle is an approximation symbol $\approx$ followed by a dollar sign $\$$.

$$H, H \cdot \text{Sparse noise} \approx \$$$

Random matrix

Sparse noise

LPN to the Rescue

The LPN assumption - dual

$$\begin{array}{c} \boxed{H} \quad , \quad \boxed{H} \cdot \text{[pink bar]} \approx \$ \\ \uparrow \qquad \qquad \qquad \uparrow \\ \text{Random matrix} \qquad \text{Sparse noise} \end{array}$$



LPN yields a simple PRG in the class:

LPN to the Rescue

The LPN assumption - dual

$$\begin{array}{c} \boxed{H} \quad , \quad \boxed{H} \cdot \begin{array}{c} \text{pink bar} \\ \uparrow \\ \text{Sparse noise} \end{array} \approx \$ \\ \uparrow \qquad \qquad \qquad \uparrow \\ \text{Random matrix} \qquad \text{Sparse noise} \end{array}$$



LPN yields a simple PRG in the class:

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

LPN to the Rescue

The LPN assumption - dual

$$\begin{array}{c} \boxed{H} \quad , \quad \boxed{H} \cdot \text{[pink vector]} \approx \$ \\ \uparrow \qquad \qquad \qquad \uparrow \\ \text{Random matrix} \qquad \text{Sparse noise} \end{array}$$



LPN yields a simple PRG in the class:

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

Linear combination

LPN to the Rescue

The LPN assumption - dual

$$\begin{array}{c} \boxed{H} \quad , \quad \boxed{H} \cdot \begin{array}{c} \text{pink bar} \end{array} \approx \$ \\ \uparrow \qquad \qquad \qquad \uparrow \\ \text{Random matrix} \qquad \text{Sparse noise} \end{array}$$

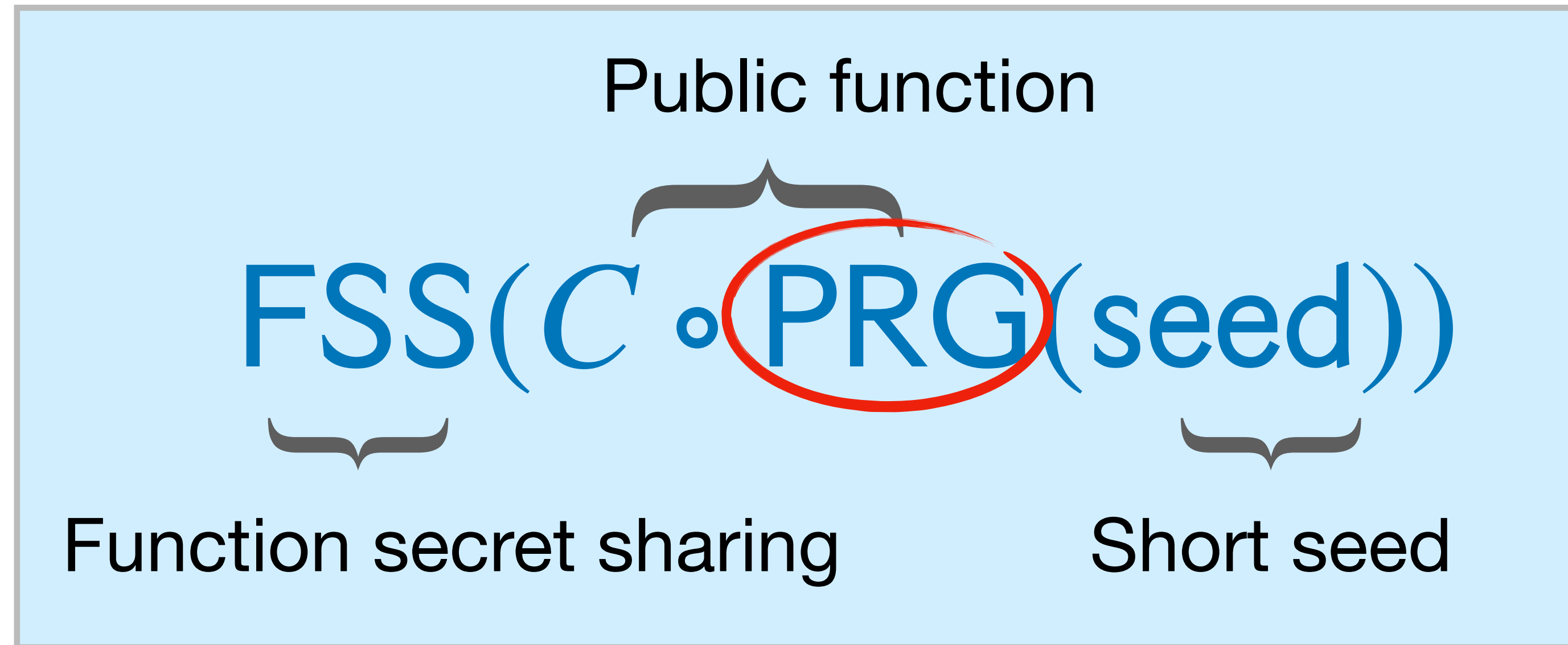


LPN yields a simple PRG in the class:

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

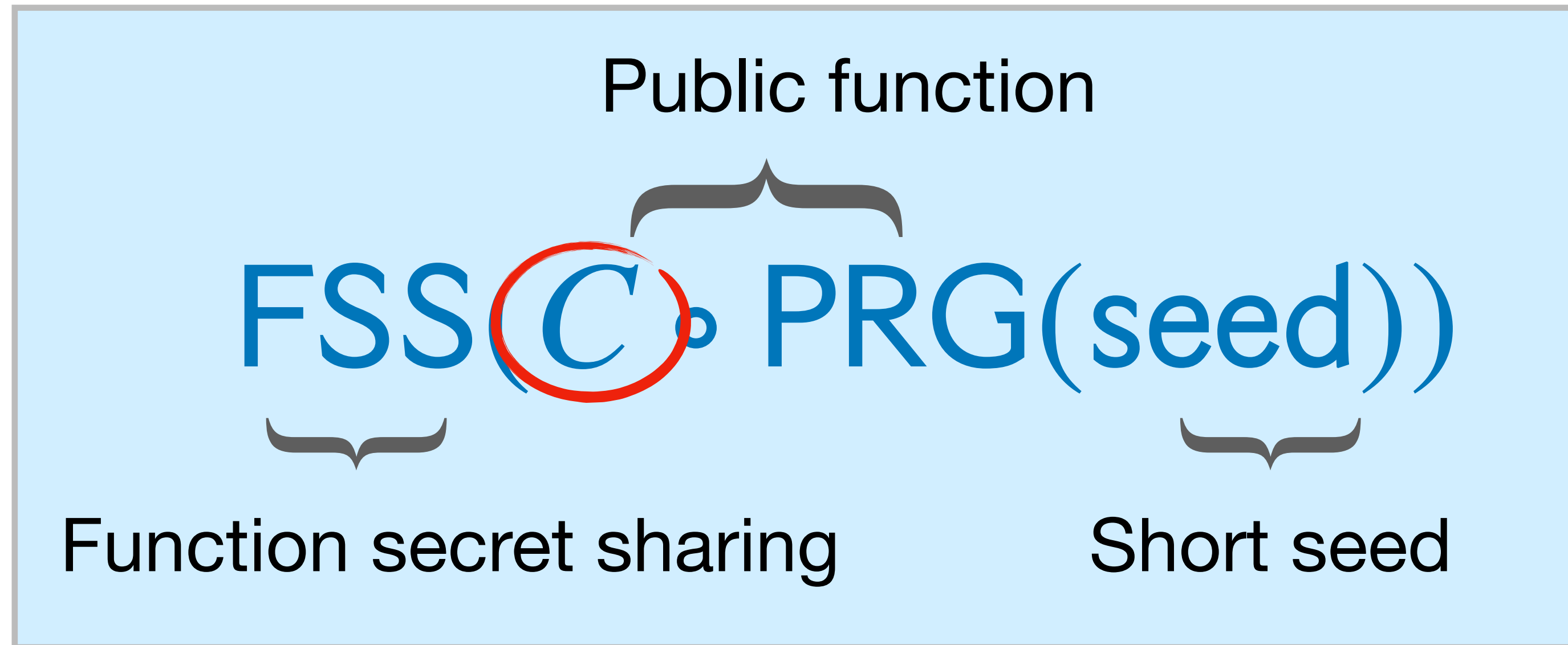
Linear combination

Back to the PCG Template Again



We have FSS for a class that contains a PRG

Back to the PCG Template Again

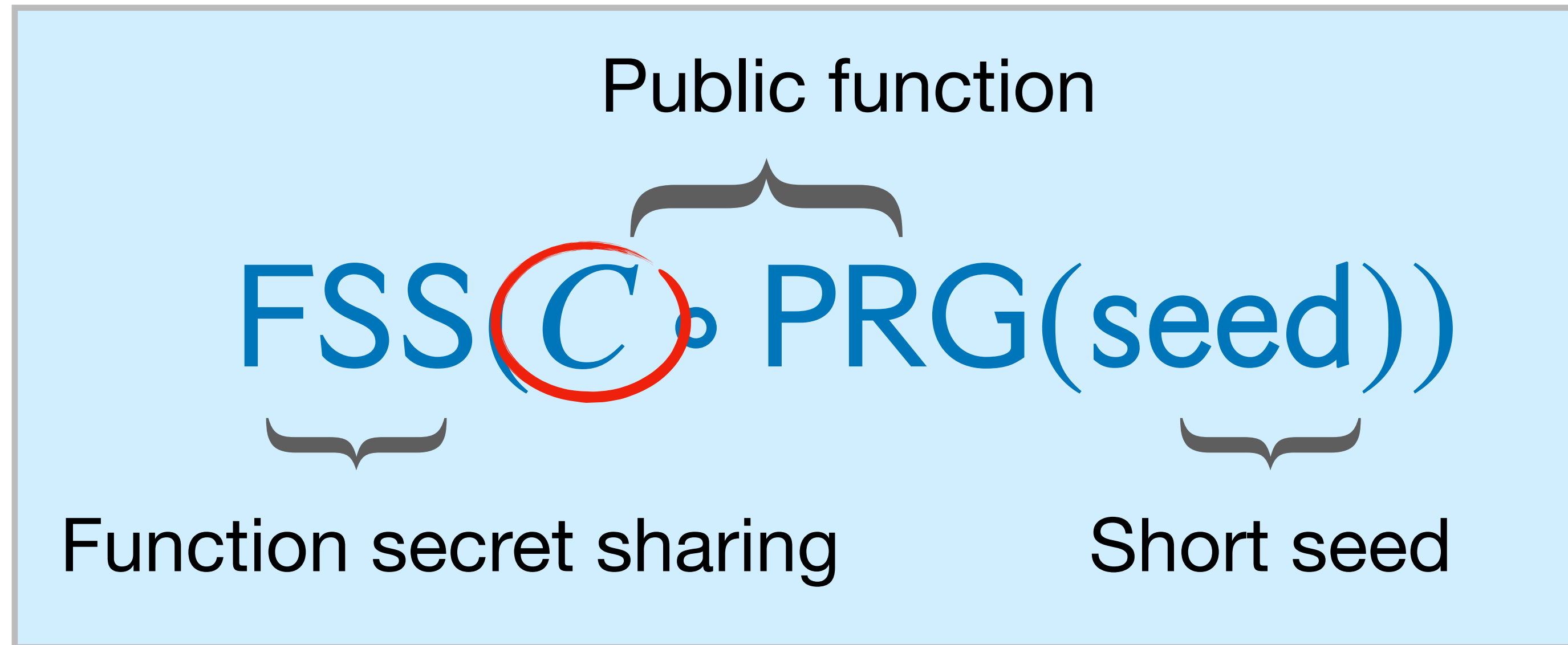


We have FSS for a class that contains a PRG

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



We have FSS for a class that contains a PRG

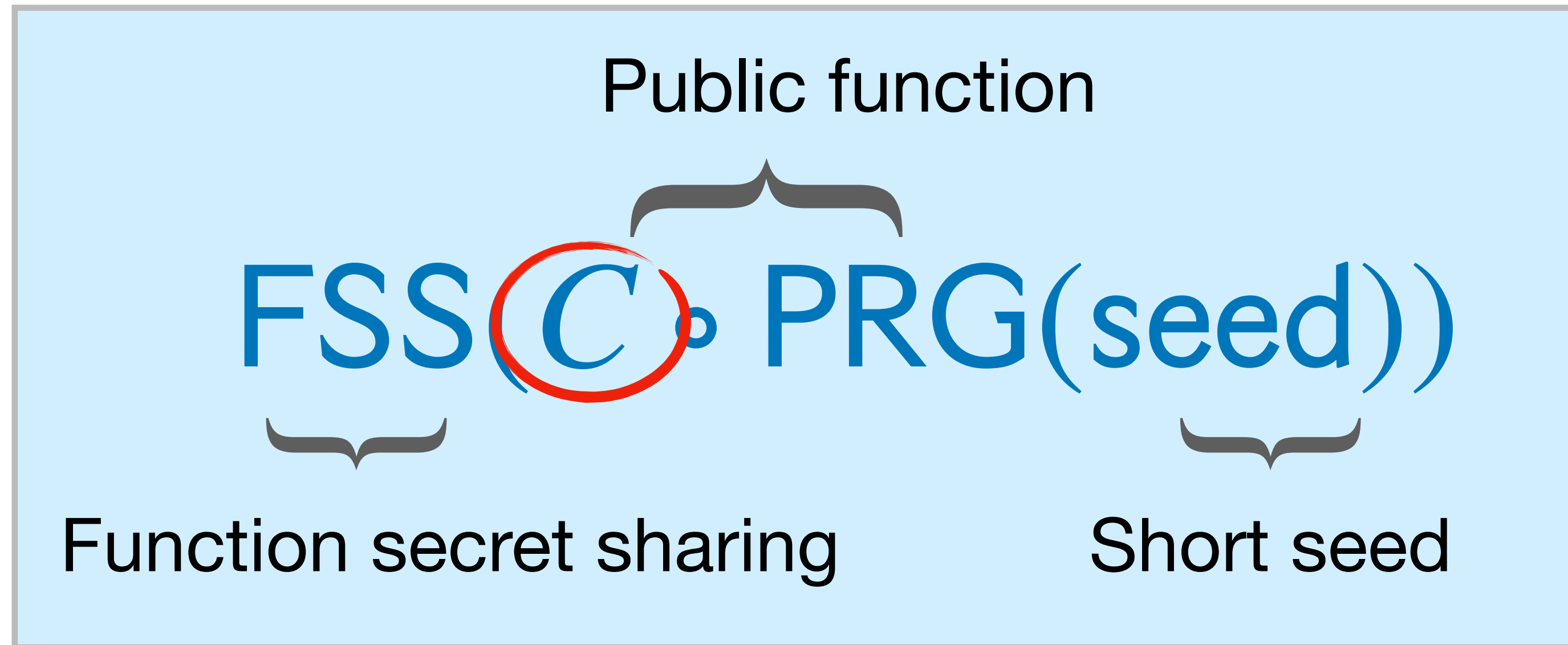
The heavy lifting in the many subsequent works boils down to:

- 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



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

$$\left(\begin{array}{c} \text{Green box } G \\ \uparrow \\ \text{Random matrix} \end{array}, \begin{array}{c} \text{Green box } G \cdot \text{Grey box} + \text{Pink box} \\ \uparrow \quad \quad \uparrow \\ \text{Short secret} \quad \text{Noise} \end{array} \right) \approx \$$$

$$\text{LPN}(\mathbb{F}_2): \text{Green box } G \leftarrow \$ \mathbb{F}_2^{m \times n}, \text{ Grey box } \leftarrow \$ \mathbb{F}_2^n, \text{ Pink box } \leftarrow \text{Ber}(\mathbb{F}_2)^n$$

'Sparse'

$$\text{LWE}(\mathbb{F}_p): \text{Green box } G \leftarrow \$ \mathbb{F}_p^{m \times n}, \text{ Grey box } \leftarrow \$ \mathbb{F}_p^n, \text{ Pink box } \leftarrow [-B, B]^n$$

'Small'

Digression: LPN versus LWE

LPN and LWE

$$\left(\begin{array}{c} \boxed{H} \\ \uparrow \text{Random matrix} \end{array}, \begin{array}{c} \boxed{H} \cdot \begin{array}{c} \boxed{} \\ \uparrow \text{Noise} \end{array} \end{array} \right) \approx \$$$

$$\text{LPN}(\mathbb{F}_2): \boxed{H} \leftarrow_{\$} \mathbb{F}_2^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ \text{Ber}(\mathbb{F}_2)^n \\ \text{'Sparse'}$$

$$\text{LWE}(\mathbb{F}_p): \boxed{H} \leftarrow_{\$} \mathbb{F}_p^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ [-B, B]^n \\ \text{'Small'}$$

Digression: LPN versus LWE

LPN and LWE

$$\left(\begin{array}{c} \boxed{H} \\ \uparrow \\ \text{Random matrix} \end{array}, \begin{array}{c} \boxed{H} \cdot \begin{array}{c} \boxed{} \\ \uparrow \\ \text{Noise} \end{array} \end{array} \right) \approx \$$$

$$\text{LPN}(\mathbb{F}_2): \boxed{H} \leftarrow_{\$} \mathbb{F}_2^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ \text{Ber}(\mathbb{F}_2)^n \\ \text{'Sparse'}$$

Statistical security

$$\text{LWE}(\mathbb{F}_p): \boxed{H} \leftarrow_{\$} \mathbb{F}_p^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ [-B, B]^n \\ \text{'Small'}$$

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

Digression: LPN versus LWE

LPN and LWE

$$\left(\begin{array}{c} \boxed{H} \\ \uparrow \text{Random matrix} \end{array}, \begin{array}{c} \boxed{H} \cdot \begin{array}{c} \boxed{} \\ \uparrow \text{Noise} \end{array} \end{array} \right) \approx \$$$

Compressibility

$$\text{LPN}(\mathbb{F}_2): \boxed{H} \leftarrow_{\$} \mathbb{F}_2^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ \text{Ber}(\mathbb{F}_2)^n \\ \text{'Sparse'}$$

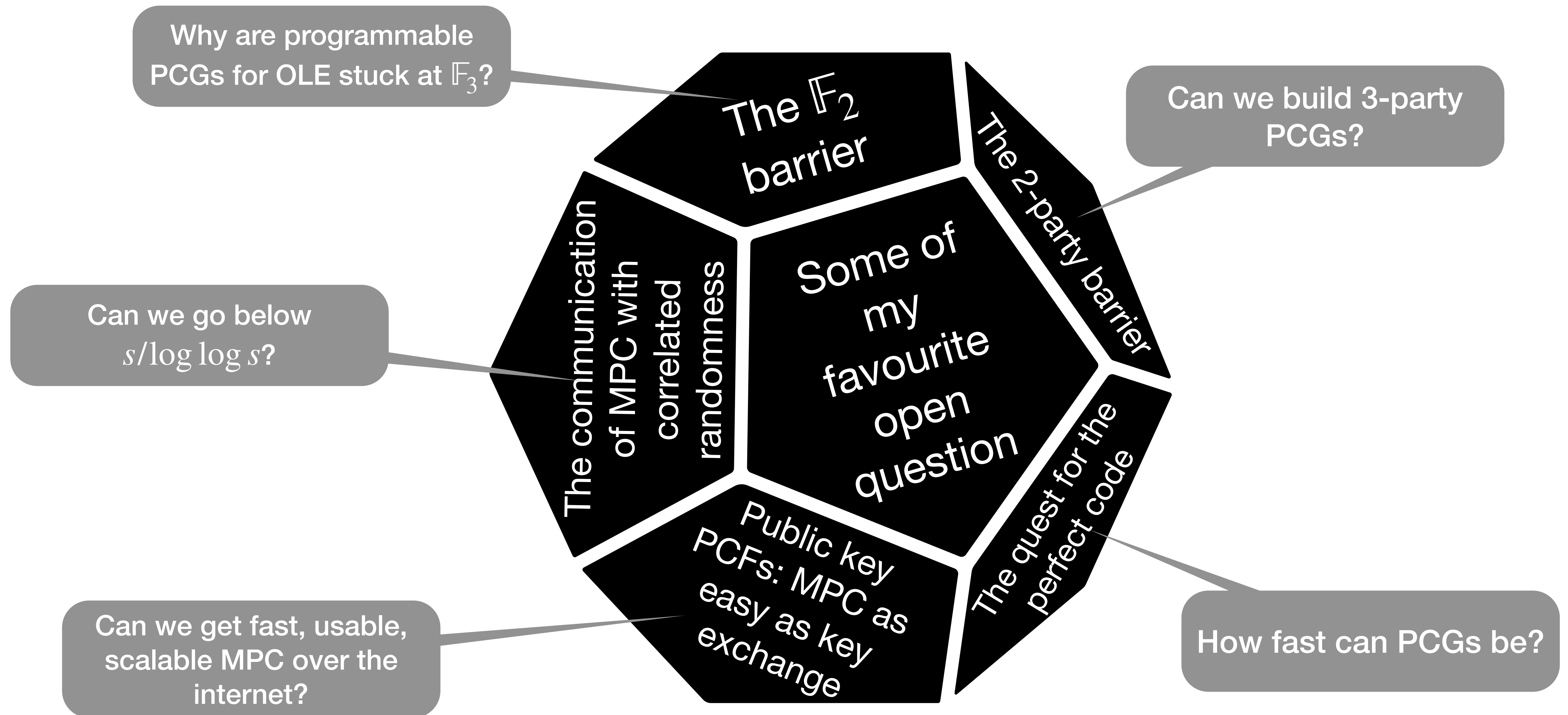
$t \cdot \log n \ll n$ entropy in the noise $\implies$ compressibility! Crucially used in recent results: PCGs, but also iO and batch OT.

Statistical security

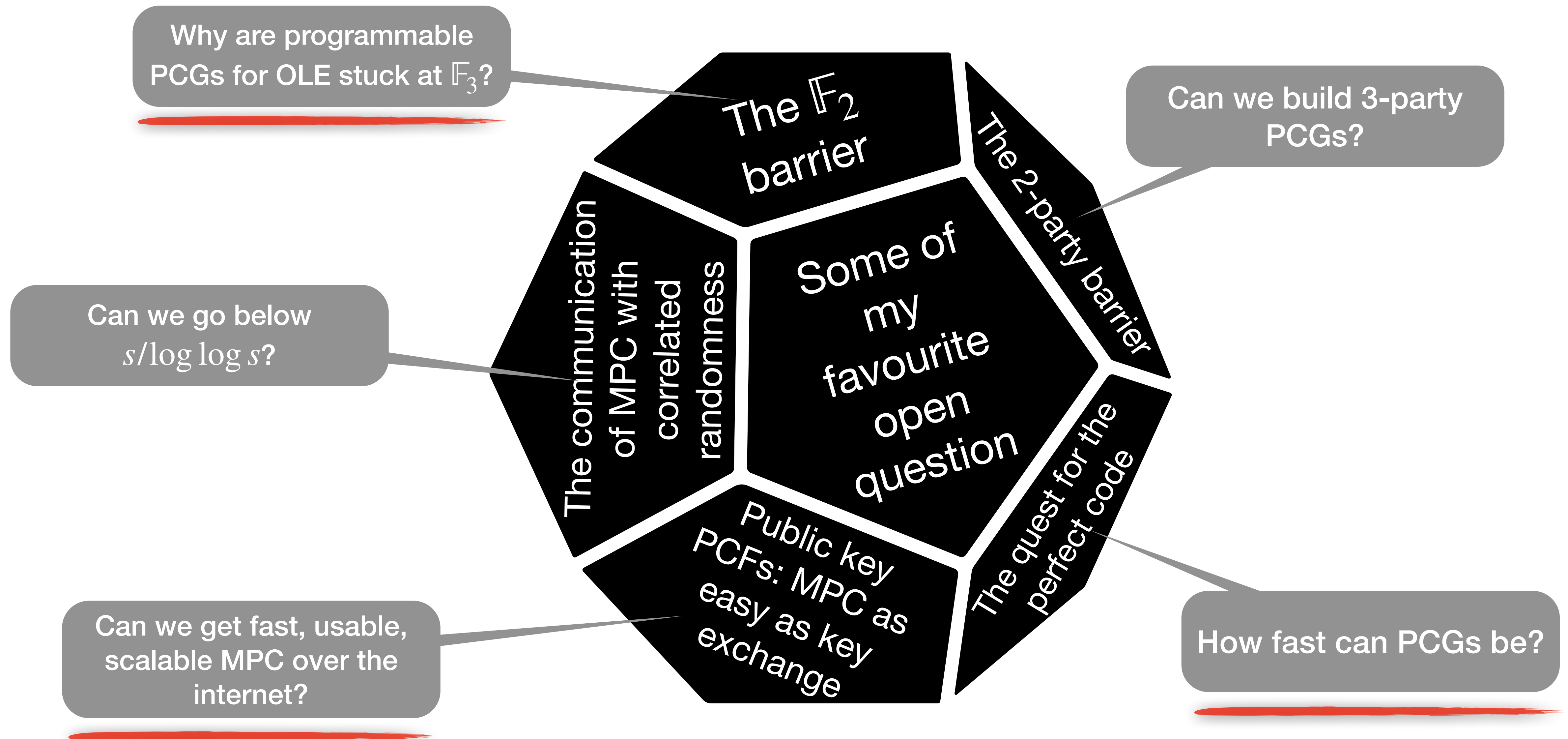
$$\text{LWE}(\mathbb{F}_p): \boxed{H} \leftarrow_{\$} \mathbb{F}_p^{m \times n}, \begin{array}{c} \boxed{} \\ \uparrow \\ [-B, B]^n \\ \text{'Small'}$$

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

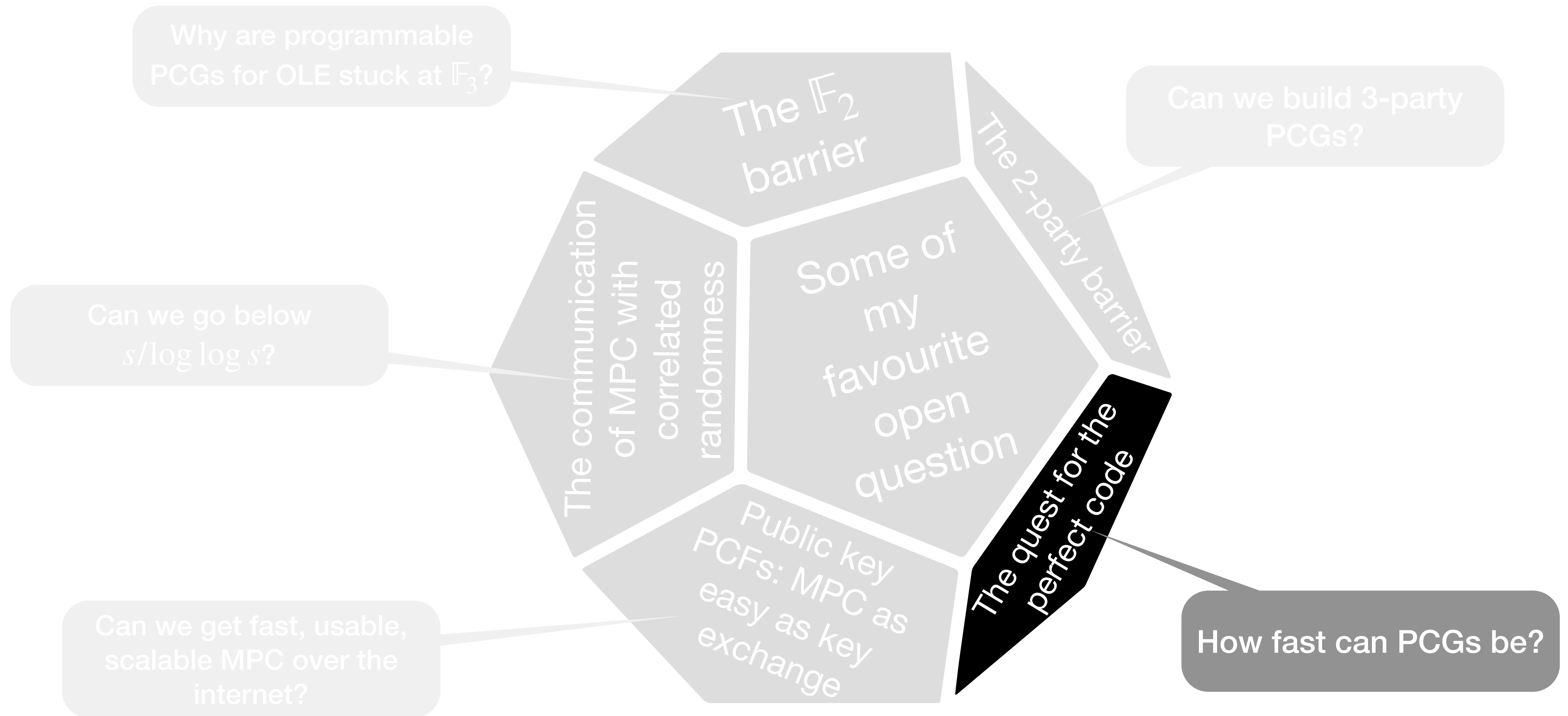
Some of my Favourite Open Questions



Some of my Favourite Open Questions



Some of my Favourite Open Questions



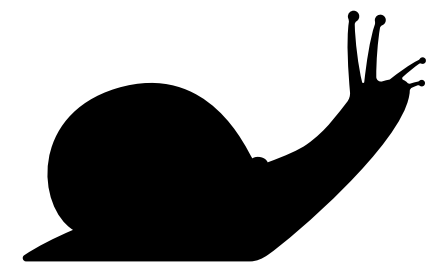
Making the PRG more Efficient

Making the PRG more Efficient

$$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto H \cdot \left(\begin{array}{c} \text{red} \\ \text{pink} \end{array} + \begin{array}{c} \text{pink} \\ \text{red} \end{array} + \begin{array}{c} \text{red} \\ \text{pink} \end{array} + \begin{array}{c} \text{pink} \\ \text{red} \end{array} \right)$$

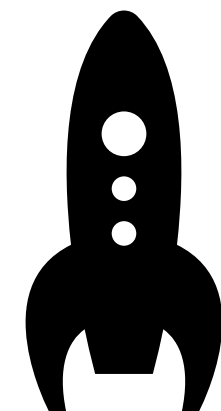
Making the PRG more Efficient

$$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto \underbrace{H}_{\text{Matrix}} \cdot \left(\underbrace{\begin{pmatrix} \text{red} \\ \text{pink} \end{pmatrix} + \begin{pmatrix} \text{pink} \\ \text{red} \end{pmatrix} + \begin{pmatrix} \text{red} \\ \text{pink} \end{pmatrix} + \begin{pmatrix} \text{pink} \\ \text{red} \end{pmatrix}}_{\text{Unit vectors}} \right)$$



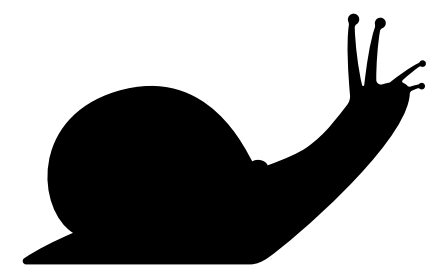
Multiplying by a random
matrix of size $\Omega(n^2)$

Generating and
summing unit vectors



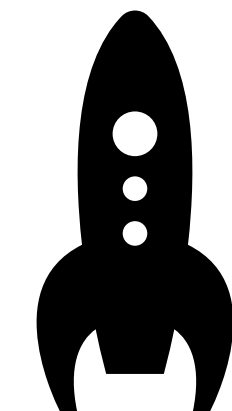
Making the PRG more Efficient

$$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto \underbrace{H}_{\text{Matrix}} \cdot \left(\underbrace{\begin{pmatrix} \text{red} \\ \text{pink} \end{pmatrix} + \begin{pmatrix} \text{pink} \\ \text{red} \end{pmatrix} + \begin{pmatrix} \text{red} \\ \text{pink} \end{pmatrix} + \begin{pmatrix} \text{pink} \\ \text{red} \end{pmatrix}}_{\text{Unit vectors}} \right)$$



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

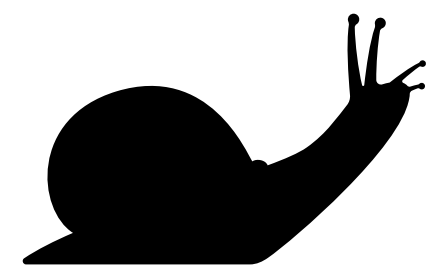
Generating and summing unit vectors



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

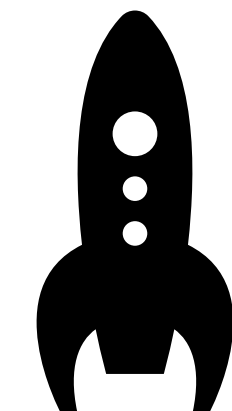
Making the PRG more Efficient

$$\text{PRG} : (\alpha_i)_{i \leq t} \mapsto \underbrace{H}_{\text{Matrix}} \cdot \left(\underbrace{\begin{pmatrix} \text{red} \\ \text{white} \\ \text{white} \end{pmatrix} + \begin{pmatrix} \text{white} \\ \text{red} \\ \text{white} \end{pmatrix} + \begin{pmatrix} \text{white} \\ \text{white} \\ \text{red} \end{pmatrix} + \begin{pmatrix} \text{white} \\ \text{white} \\ \text{white} \\ \text{red} \end{pmatrix}}_{\text{Unit vectors}} \right)$$



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: $n \sim 2^{30}$)

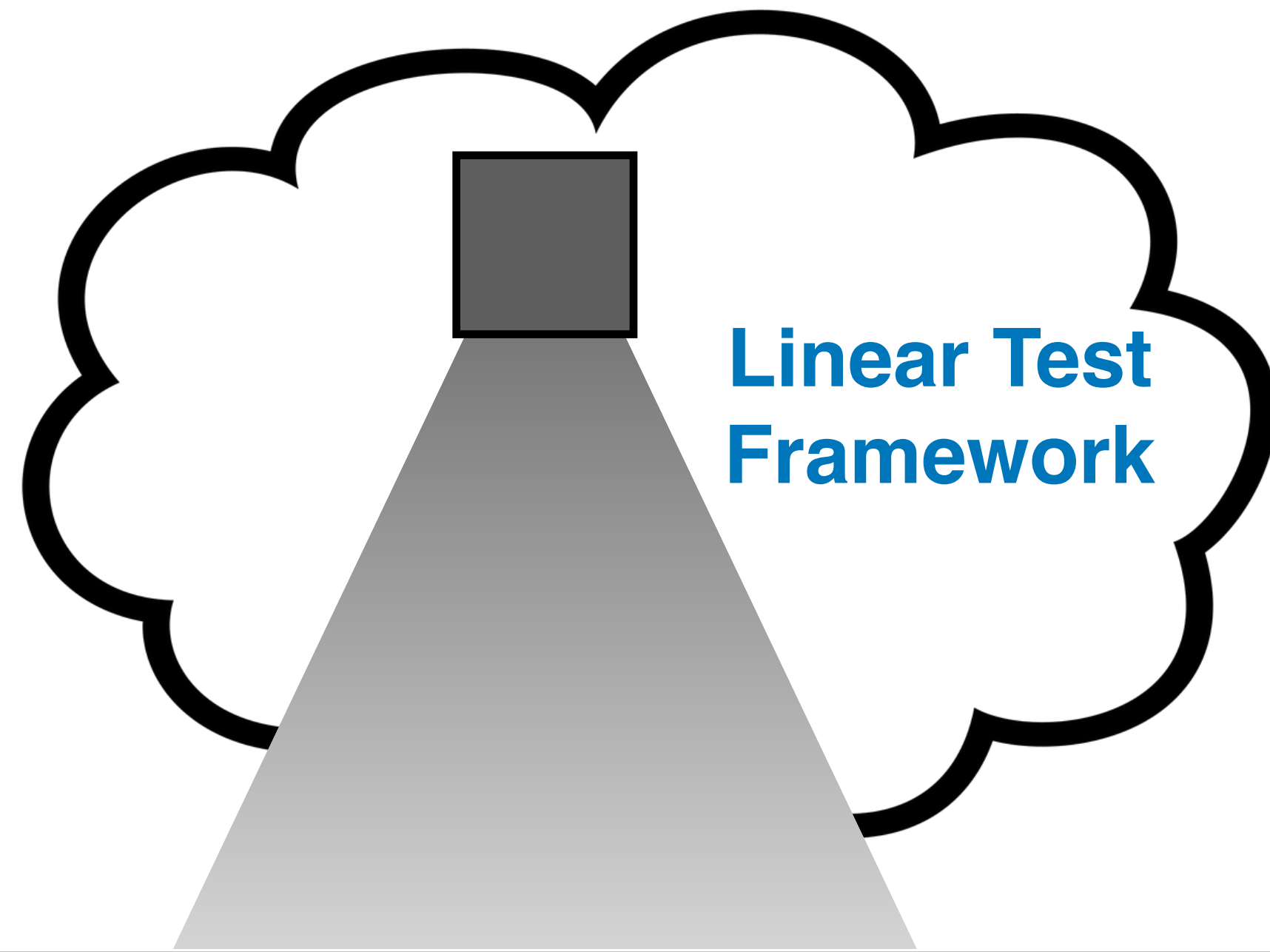


Can we replace H with a matrix that allows for fast matrix-vector product?

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

Security of (variants of) LPN - Linear Tests

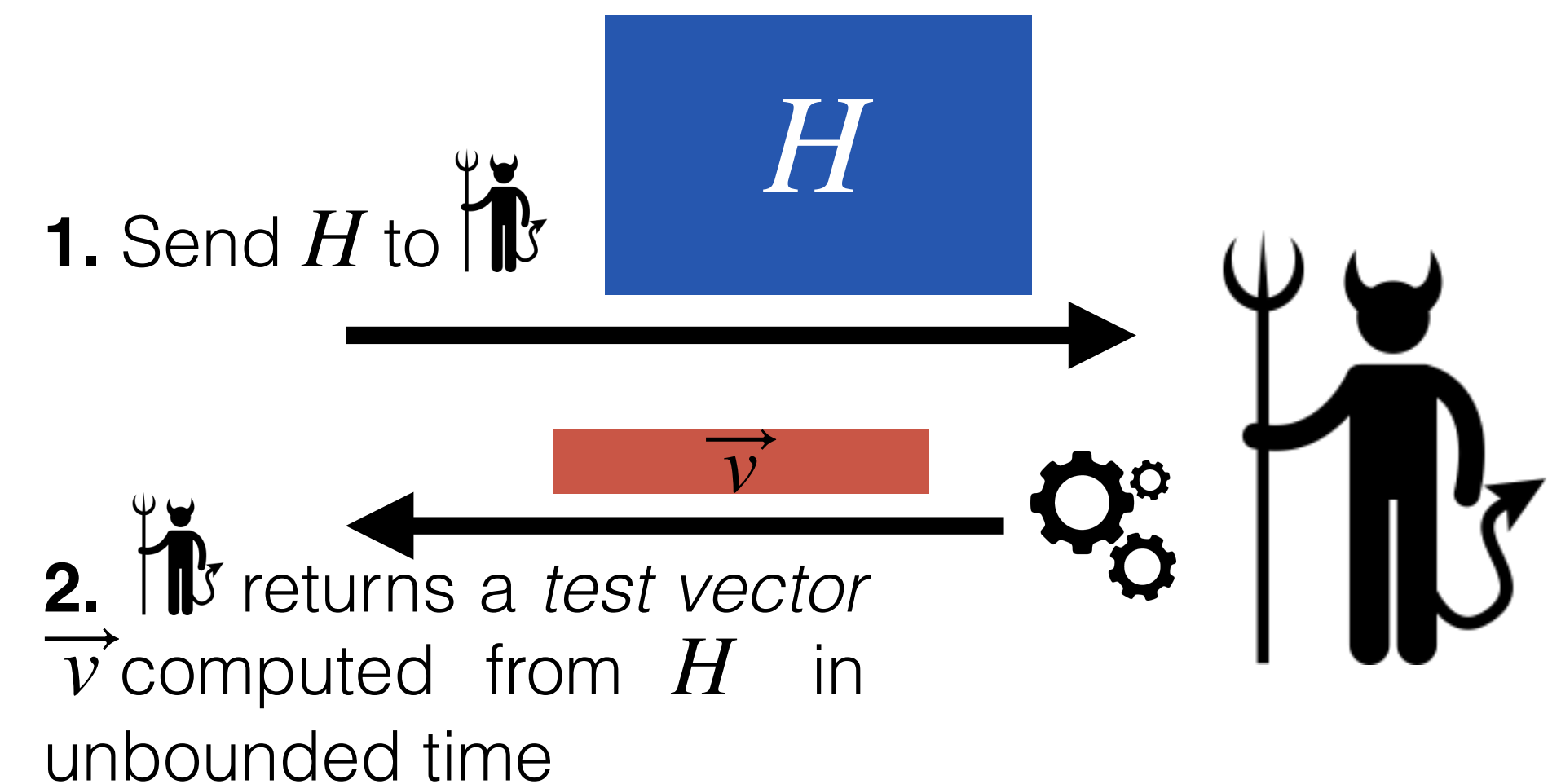
A tremendous number of attacks on LPN have been published...



- **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]
 - Levie-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



Check

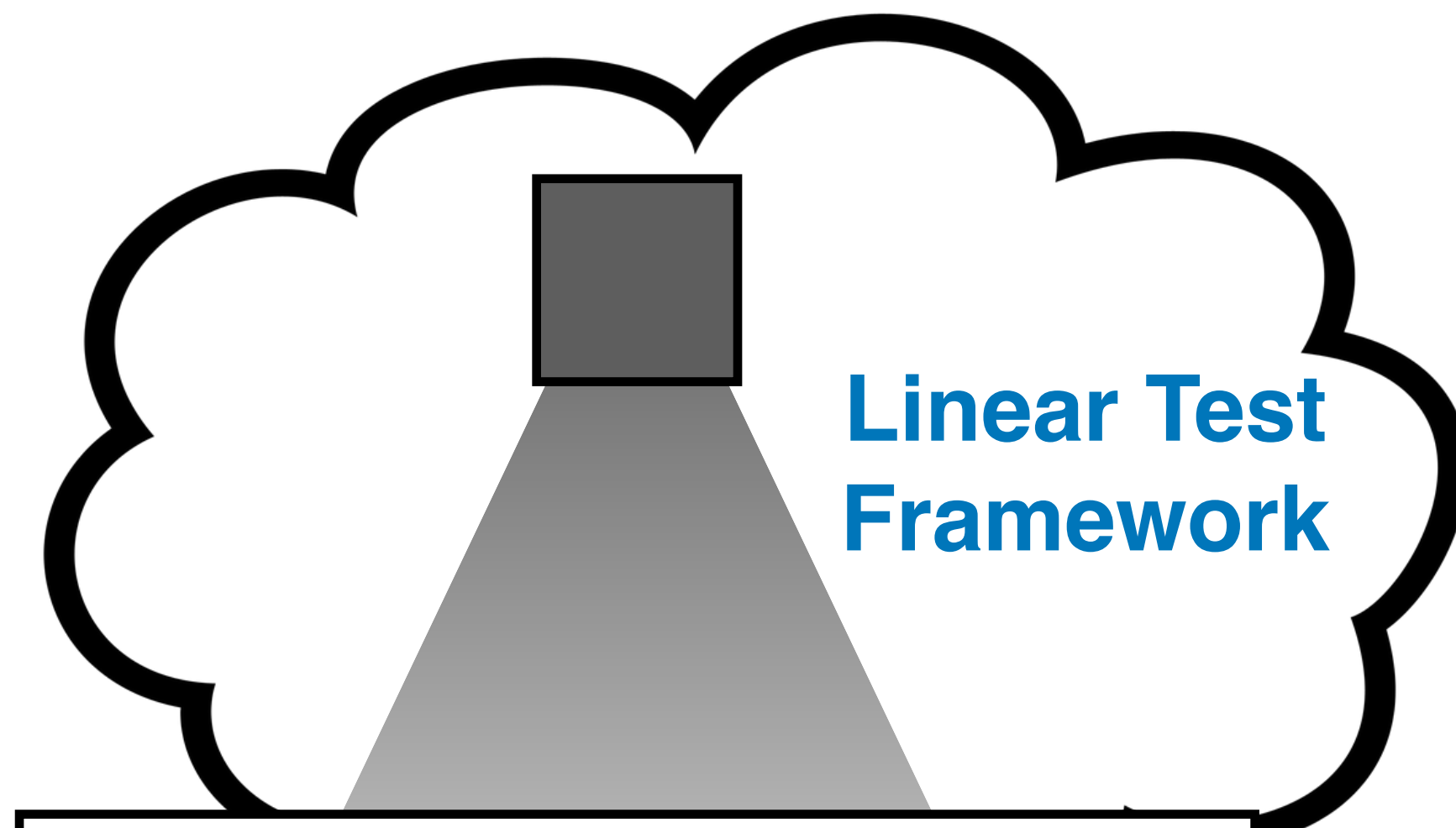
The adversary wins in the distribution induced by

$$\text{Red Box} \cdot \left(\text{Blue Box } H \cdot \text{Pink Box} \right)$$

(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...

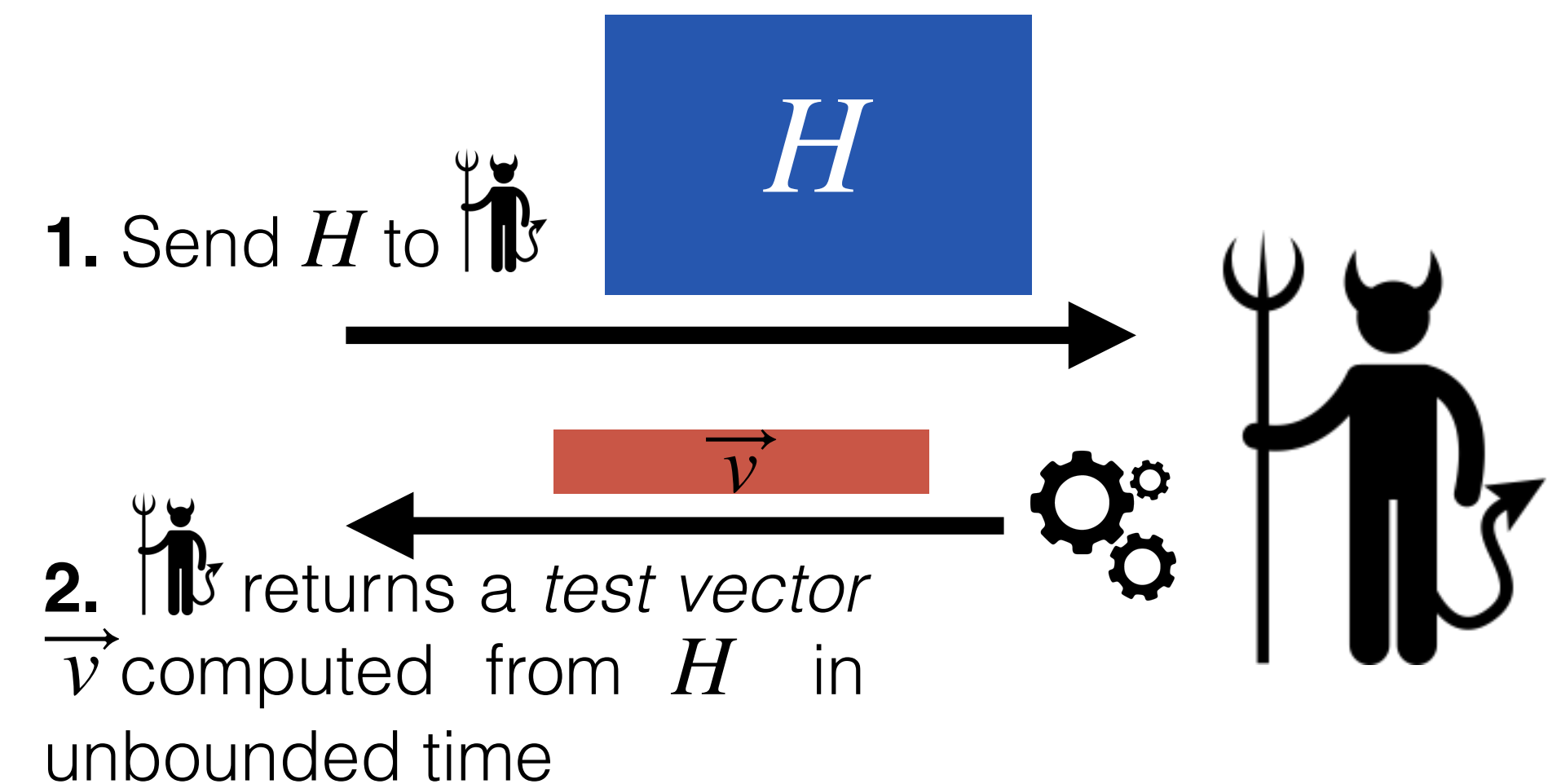


(*): highly structured algebraic codes (e.g. Reed-Solomon) are a different beast

- **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]
 - Levie-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



Check

The adversary wins in the distribution induced by

$$\text{Red Bar} \cdot \left(H \cdot \text{Pink Bar} \right)$$

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

Withstanding Linear Tests

The adversary wins in the distribution induced by

$$\overrightarrow{v} \cdot \left(G \cdot \text{secret} + \text{noise} \right)$$

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

We have a sum of two distributions:

Induced by the *codeword*

$$\overrightarrow{v} \cdot G \cdot \text{secret}$$

Protects against *light* linear tests

Induced by the *noise vector*

$$\overrightarrow{v} \cdot \text{noise}$$

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

$$\overrightarrow{v} \cdot \left(G \cdot \text{secret} + \text{noise} \right)$$

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

We have a sum of two distributions:

Induced by the *codeword*

$$\overrightarrow{v} \cdot G \cdot \text{secret}$$

Protects against *light* linear tests

Induced by the *noise vector*

$$\overrightarrow{v} \cdot \text{noise}$$

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

$\iff$ the left-kernel of G does not contain nonzero vector of weight less than $O(n)$

$\iff$ 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-friendly

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

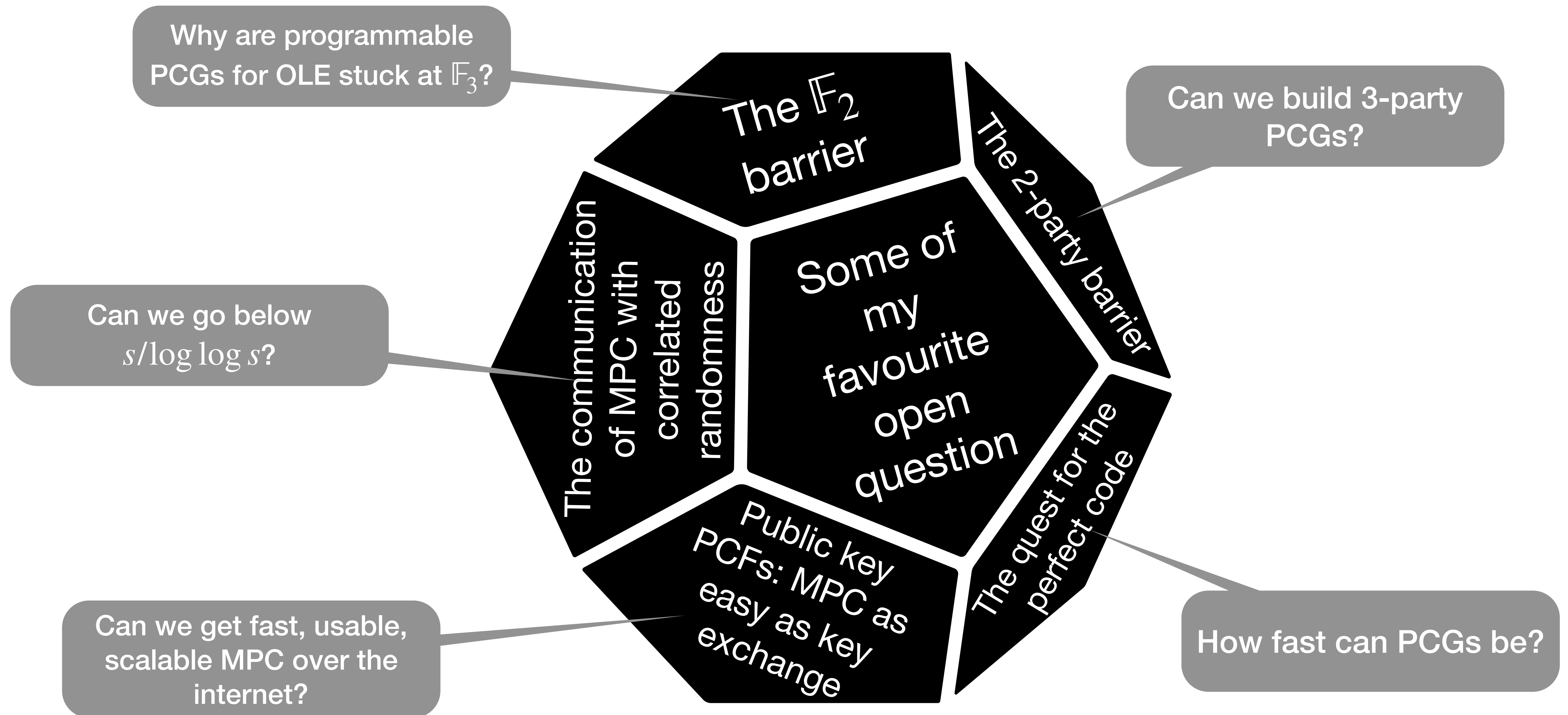
$\Rightarrow$ We need to find a *good* and *linear-time encodable* code. And we want it concretely efficient!

Pseudorandom Correlation Generators - Efficiency

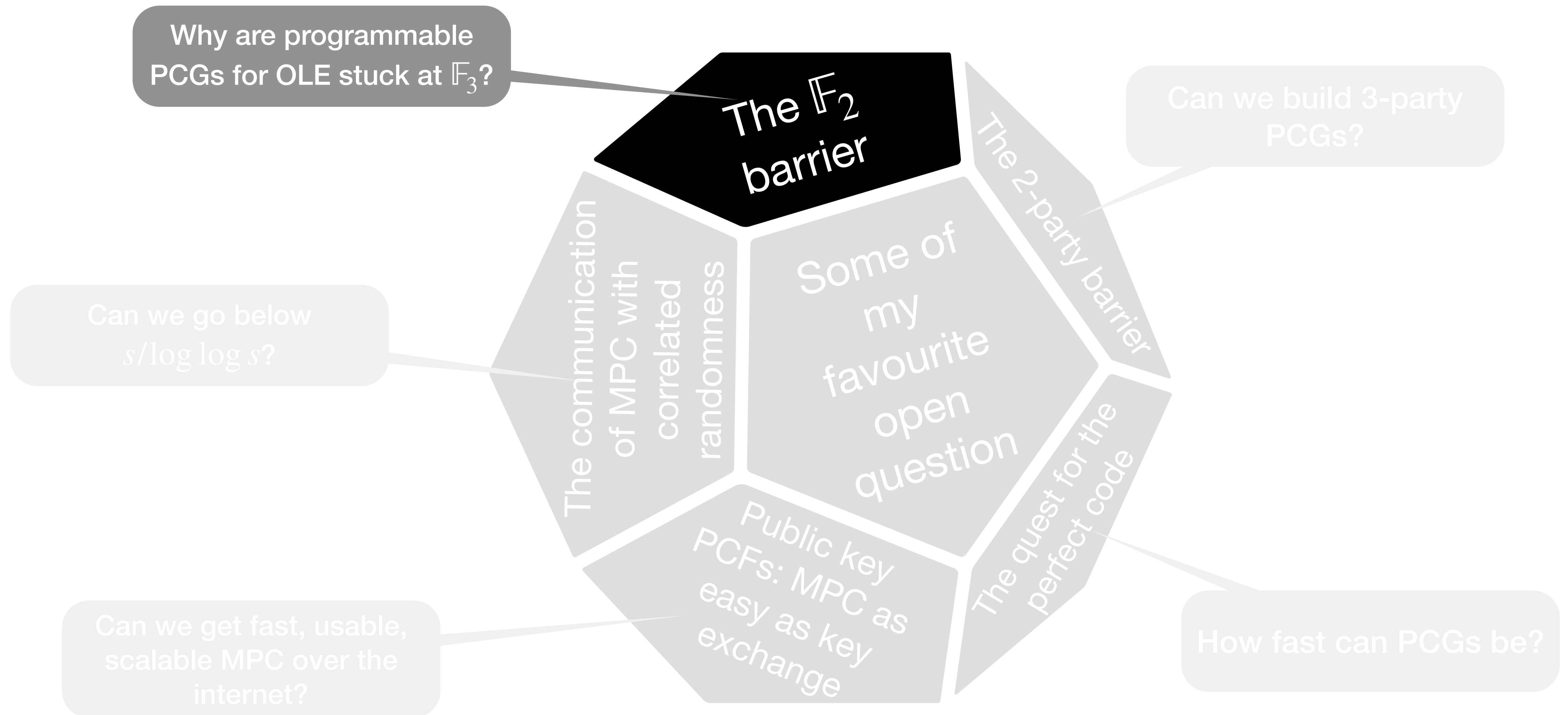
There is an ongoing and exciting quest for pinpointing the *right* code for PCG applications:

- 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 Questions



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

The diagram illustrates the dot product of two vectors, $\vec{x}$ and $\vec{y}$. On the left, a vertical blue rectangle represents vector $\vec{x}$, with the label $\vec{x}$ at its bottom. To its right is a dot operator $\cdot$. Further right is a horizontal blue rectangle representing vector $\vec{y}$, with the label $\vec{y}$ at its center. To the right of this is an equals sign $=$. Finally, on the right, is a large blue square representing the result of the dot product, with the label $\vec{x} \cdot \vec{y}^\top$ in its center.

$$\vec{x} \cdot \vec{y} = \vec{x} \cdot \vec{y}^\top$$

OLE Correlations, the LPN Way

Goal:

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

The diagram illustrates the goal of OLE correlations using the LPN way. It shows the dot product of vectors $\vec{x}$ and $\vec{y}$ resulting in a scalar value, which is then used to generate shares of the inner product.

On the left, a vertical blue bar represents vector $\vec{x}$. To its right is a horizontal blue bar representing vector $\vec{y}$. A black dot $\cdot$ is placed between them, followed by an equals sign $=$. To the right of the equals sign is a large blue square representing the result of the dot product, $\vec{x} \cdot \vec{y}^T$. A red oval is drawn around the blue square, and a red arrow points from the expression $\vec{x} \odot \vec{y}$ (written in red above the oval) to the blue square.

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 $\vec{x} \odot \vec{y}$

The diagram illustrates the goal of OLE correlations using the LPN way. It shows the dot product of vectors $\vec{x}$ and $\vec{y}$ resulting in a scalar value, which is then used to generate shares.

On the left, a vertical blue rectangle is labeled $\vec{x}$. To its right is a dot $\cdot$, followed by a horizontal blue rectangle labeled $\vec{y}$. To the right of this is an equals sign $=$, followed by a large blue square. Inside the square, the expression $\vec{x} \cdot \vec{y}^T$ is written. A red oval encircles the blue square. Above the oval, the expression $\vec{x} \odot \vec{y}$ is written in red.

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 $\vec{x} \odot \vec{y}$

$$\begin{array}{c} \vec{x} \end{array} \cdot \vec{y} = H \cdot \begin{array}{c} \vec{e}_x \end{array} \cdot \vec{e}_y^T \cdot H^T$$

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 $\vec{x} \odot \vec{y}$

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

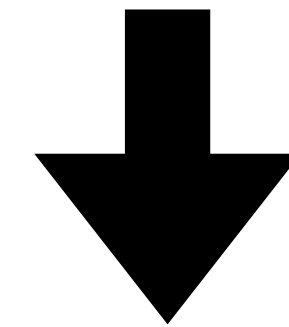
The diagram illustrates the LPN way for OLE correlations. It shows the dot product of vectors $\vec{x}$ and $\vec{y}$, which is equal to the dot product of $H \cdot \vec{e}_x$ and $H \cdot \vec{e}_y$. This is further shown as the dot product of $\vec{e}_x$ and $H^T \cdot H \cdot \vec{e}_y$.

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



The diagram illustrates the generation of pseudorandom vectors $\vec{x}$ and $\vec{y}$ using a matrix H and basis vectors $\vec{e}_x$ and $\vec{e}_y$. It shows the following components and their relationships:

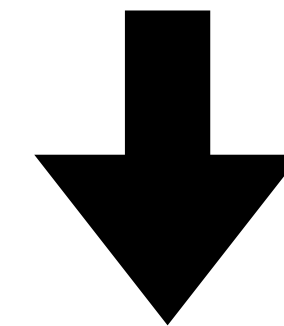
- A vertical blue bar on the left is labeled $\vec{x}$.
- A horizontal blue bar is labeled $\vec{y}$.
- An equals sign ($=$) is placed between the blue bars and the first green matrix.
- A green square matrix is labeled H .
- A red square matrix is labeled $\vec{e}_x \cdot \vec{e}_y^\top$.
- A green square matrix is labeled $H^\top$.
- Dot products ($\cdot$) are indicated between the blue bars and the green matrices, and between the red matrix and the second green matrix.

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



$$\begin{array}{c} \vec{x} \end{array} \cdot \begin{array}{c} \vec{y} \end{array} = \begin{array}{c} H \end{array} \cdot \begin{array}{c} \vec{e}_x \cdot \vec{e}_y^\top \end{array} \cdot \begin{array}{c} H^\top \end{array}$$



Uses only LPN



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

OLE Correlations, the *Ring*-LPN Way

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

Let $\mathcal{R}$ 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 $(a, b) \leftarrow \mathcal{R}$ 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 $\mathcal{R}$: 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 $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t -sparse polynomials over $\mathcal{R}$
- 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 $\mathcal{R}$ 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 $(a, b) \leftarrow \mathcal{R}$ 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 $\mathcal{R}$: 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 $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t -sparse polynomials over $\mathcal{R}$
- 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 $\mathcal{R}$ 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 $(a, b) \leftarrow \mathcal{R}$ 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 $\mathcal{R}$: 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 $\mathcal{R}$
- Bob gets a pseudorandom vector $y = a \cdot e_y + f_y$ where (e_y, f_y) are t -sparse polynomials over $\mathcal{R}$
- 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 $\mathcal{R} = \mathbb{F}[X_1, \dots, X_d]/P(X_1, \dots, X_d)$ such that

- $\mathcal{R} \sim \mathbb{F} \times \dots \times \mathbb{F}$ (*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

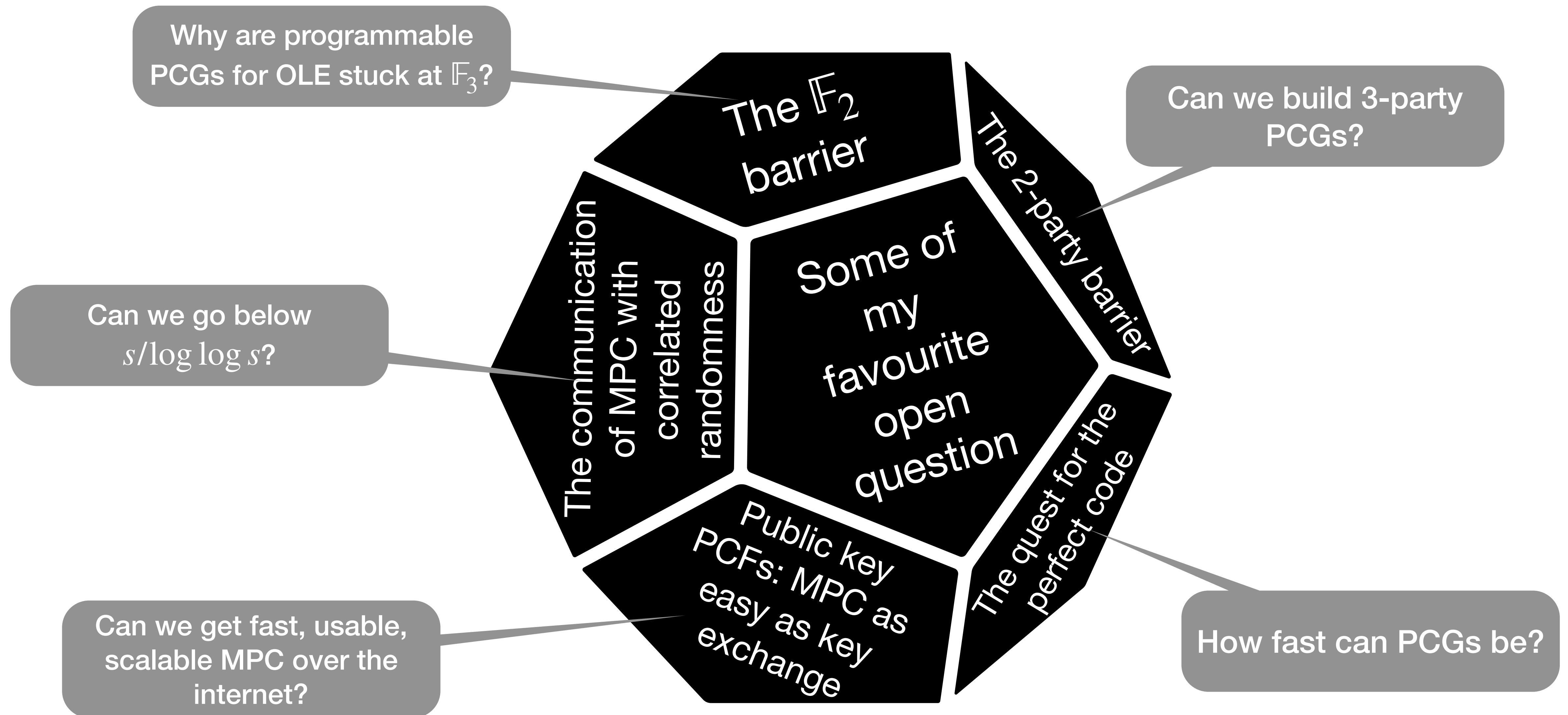
The group algebra structure gives a suitable framework to find the *right* polynomial P to instantiate an LPN variant over a ring $\mathcal{R} = \mathbb{F}[X_1, \dots, X_d]/P(X_1, \dots, X_d)$ such that

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

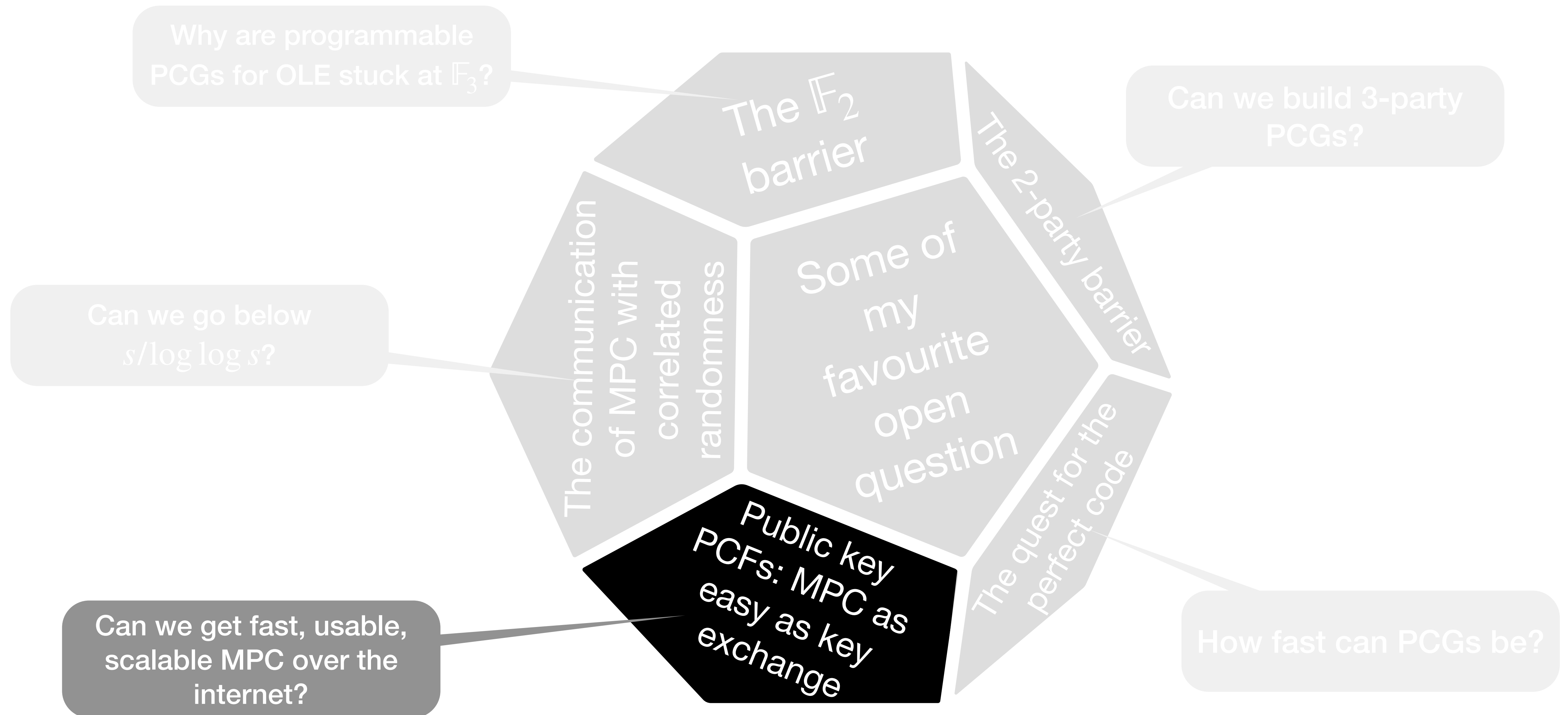
Using multivariate rings gives us many more roots of P even for a small $\mathbb{F}$! In fact, we can get up to $(|\mathbb{F}| - 1)^d$ copies of an OLE over $\mathbb{F}$.

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

Some of my Favourite Open Questions



Some of my Favourite Open Questions



A Closer Look at Secure Communication

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

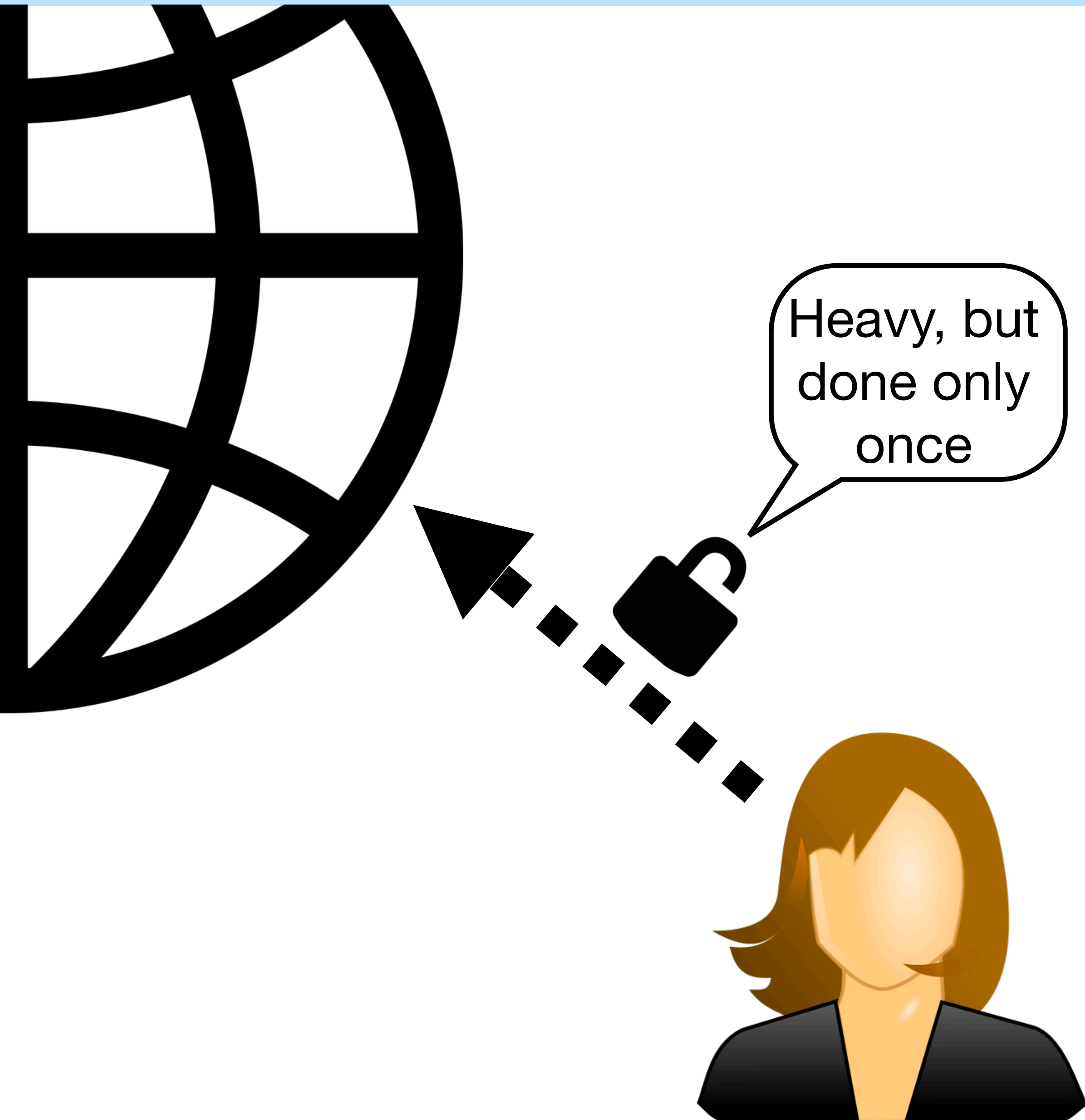
Secure communication is already widely deployed and in use



> 85% of the total internet traffic is encrypted

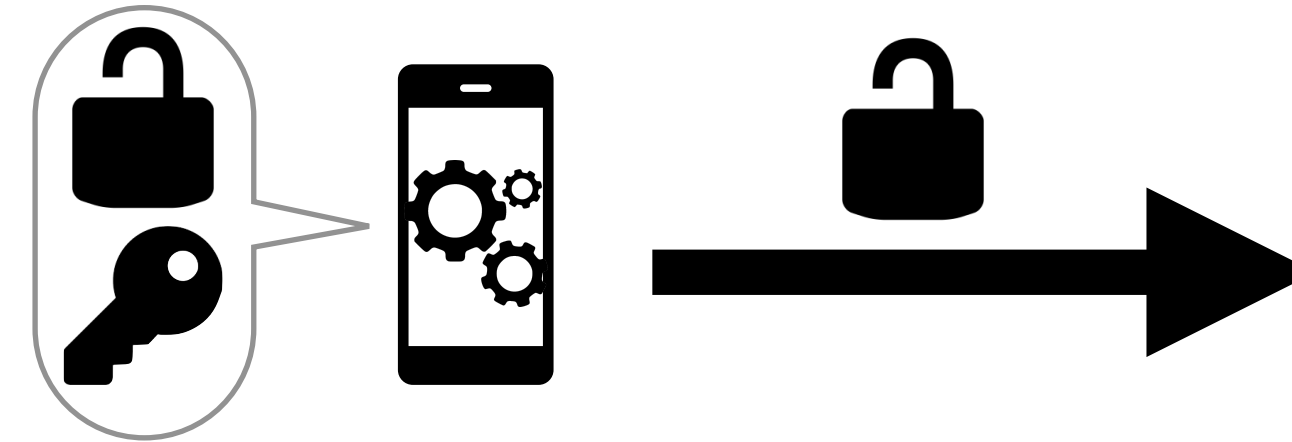
⇒ Let us look at secure communication's recipe for success!

A Closer Look at Secure Communication



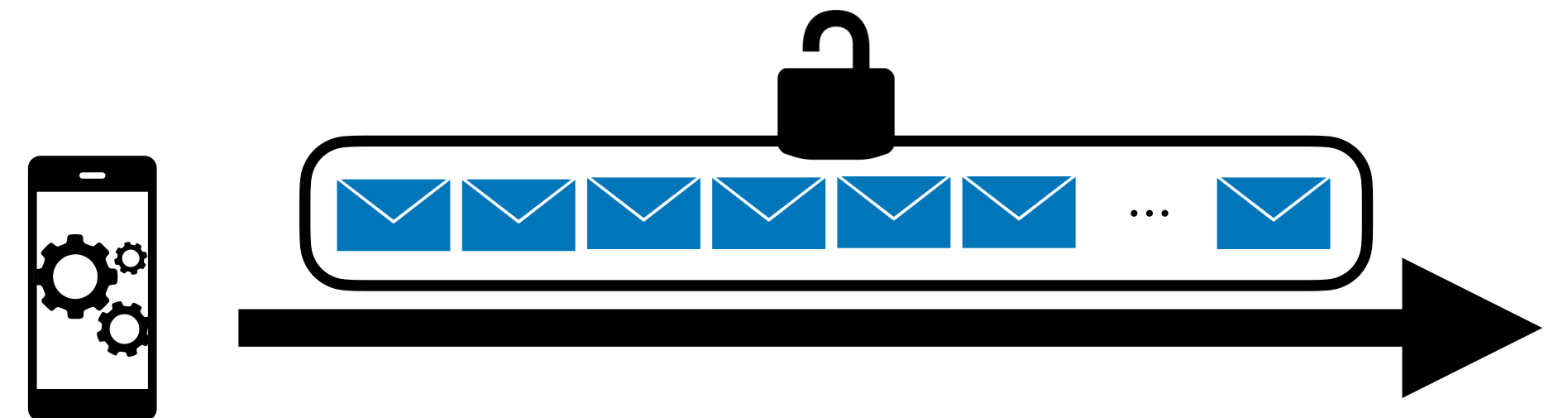
Two Phases:

Key exchange phase



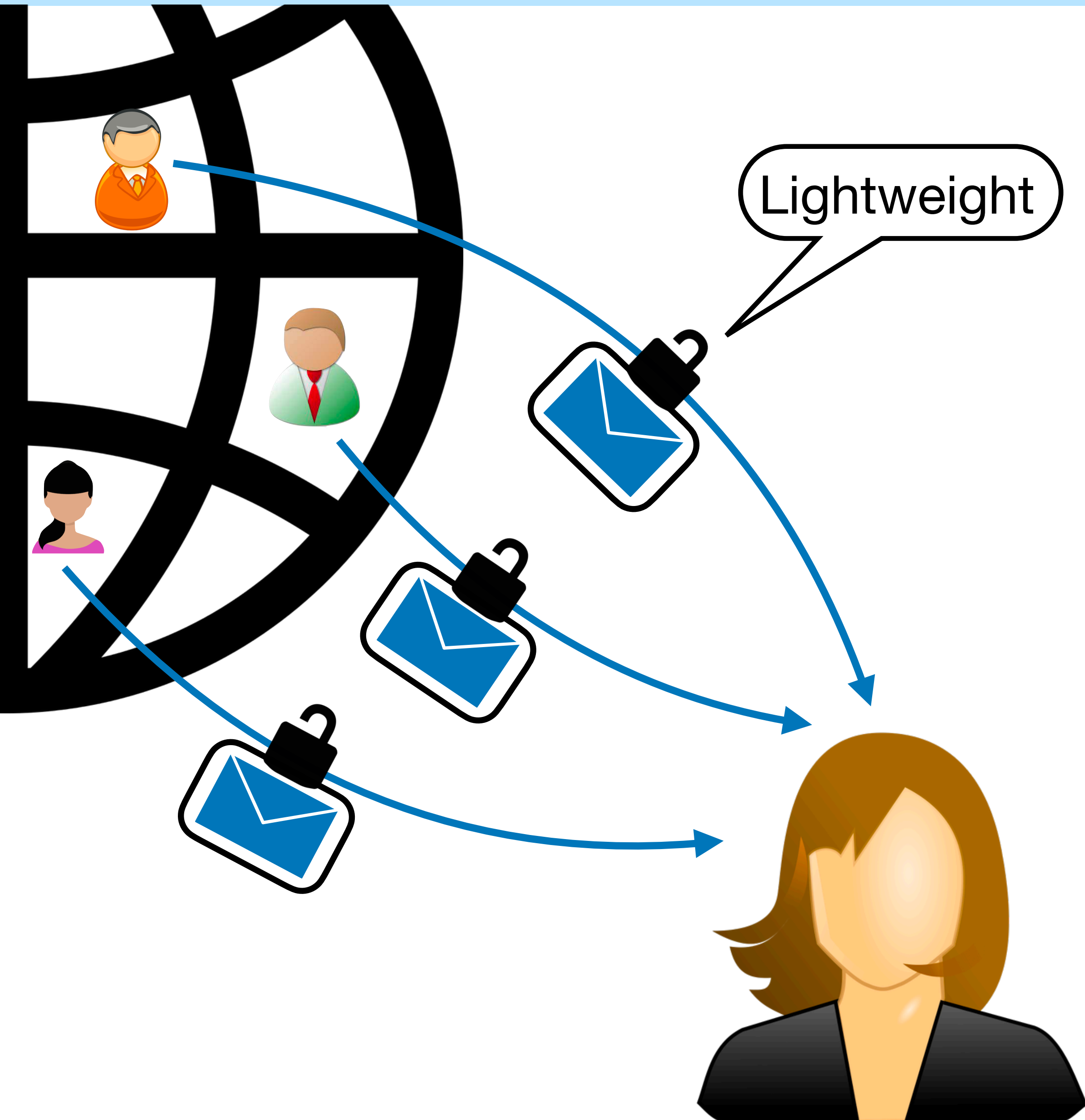
- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot |\text{padlock}|$ (not n^2)

Encryption phase



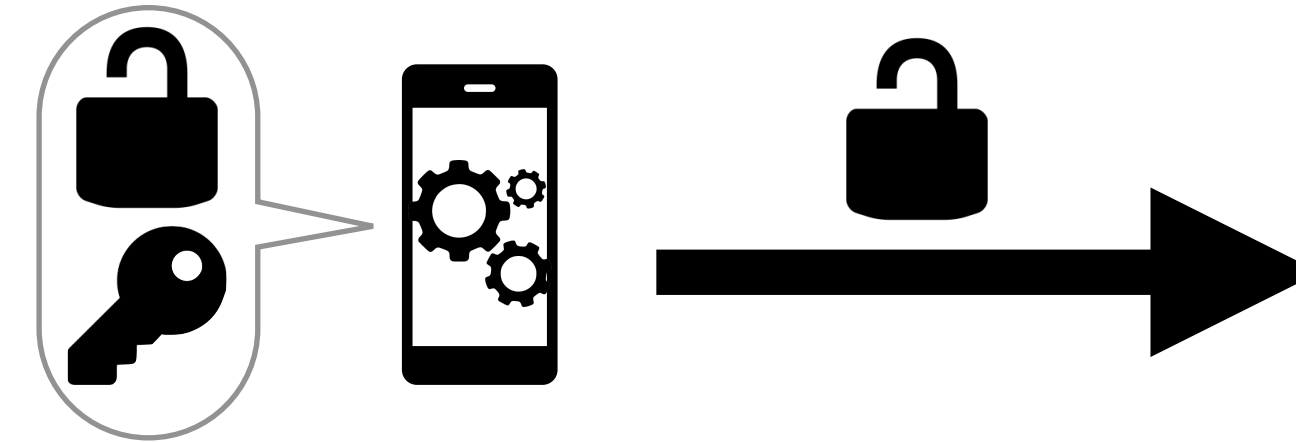
- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio

A Closer Look at Secure Communication



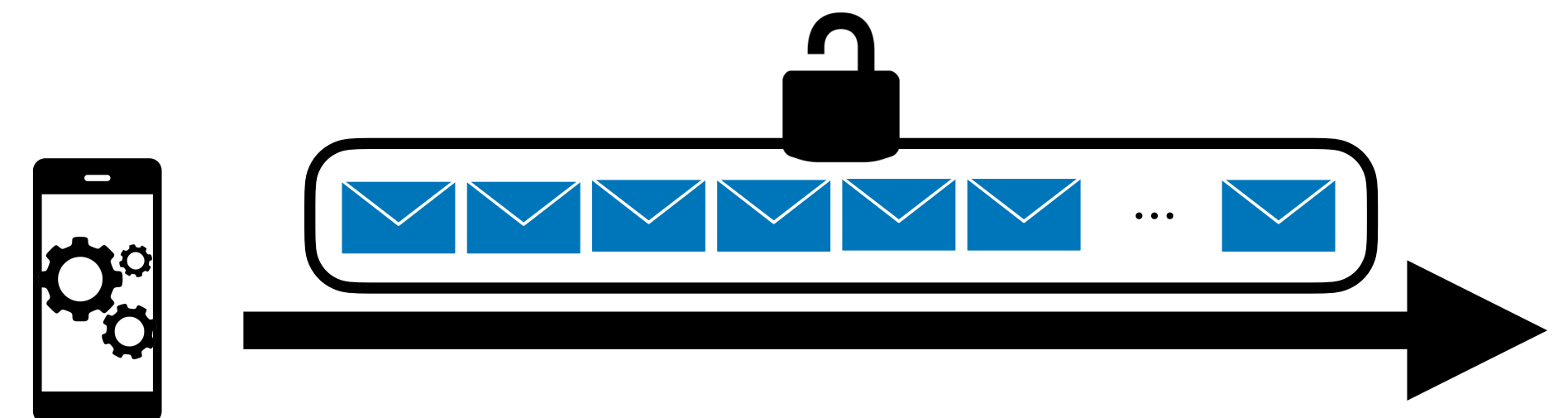
Two Phases:

Key exchange phase



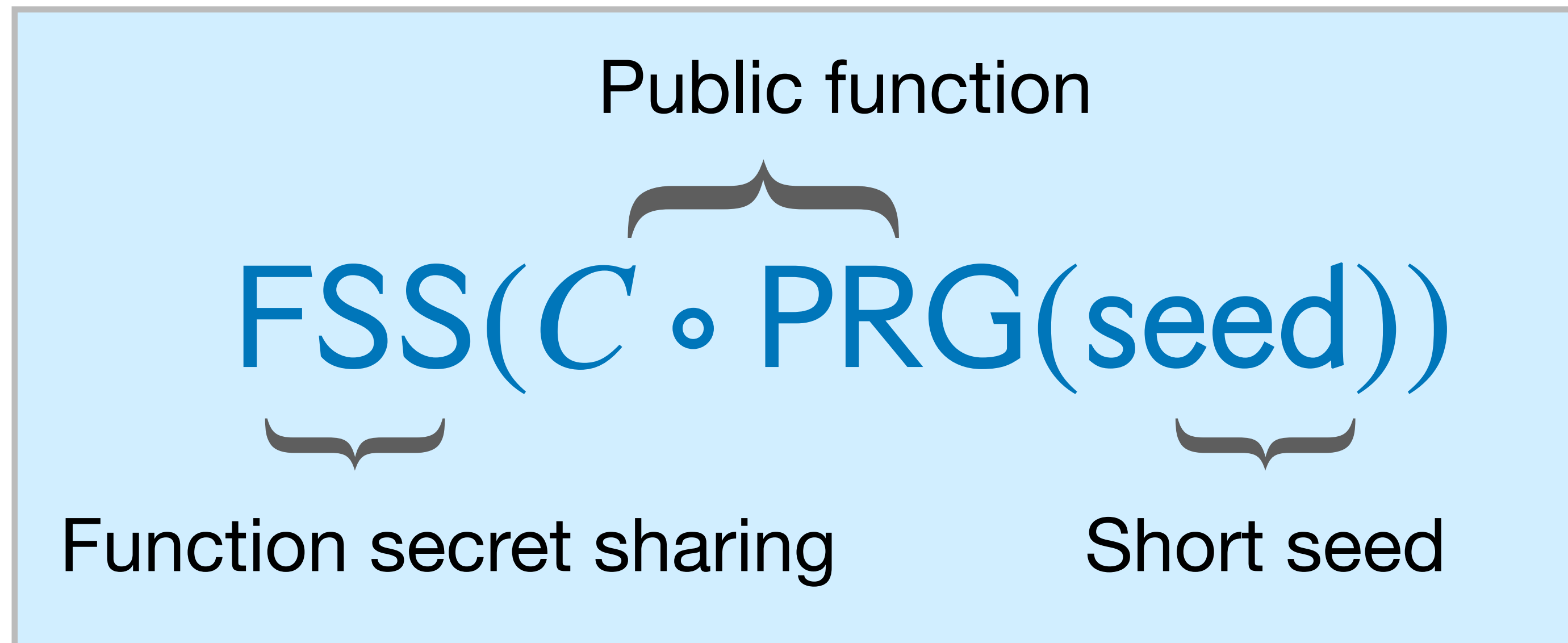
- One-time, *simultaneous* interaction
- Heavy (public key) computations
- Low communication $n \cdot |\text{padlock}|$ (not n^2)

Encryption phase



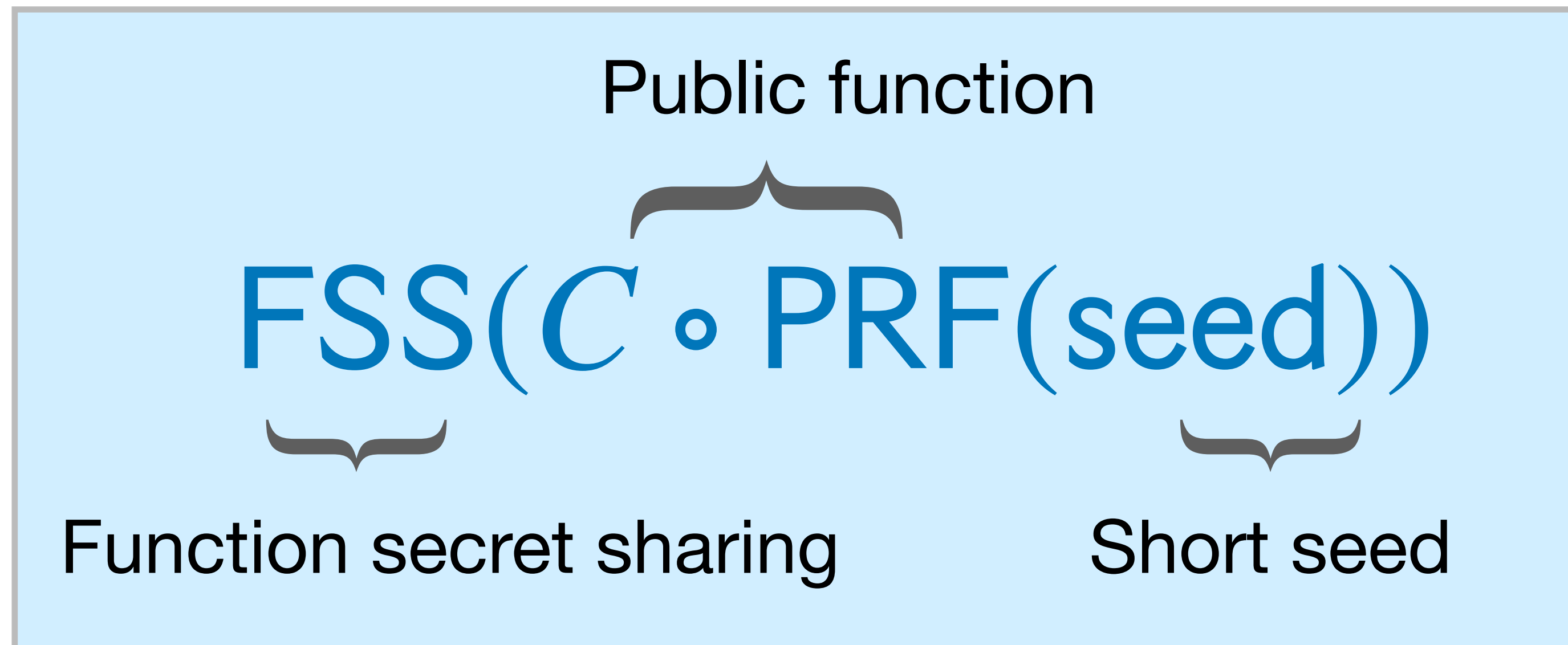
- Lightweight (symmetric) computations
- Optimal message-to-cipher ratio

Back to the PCG Template Again



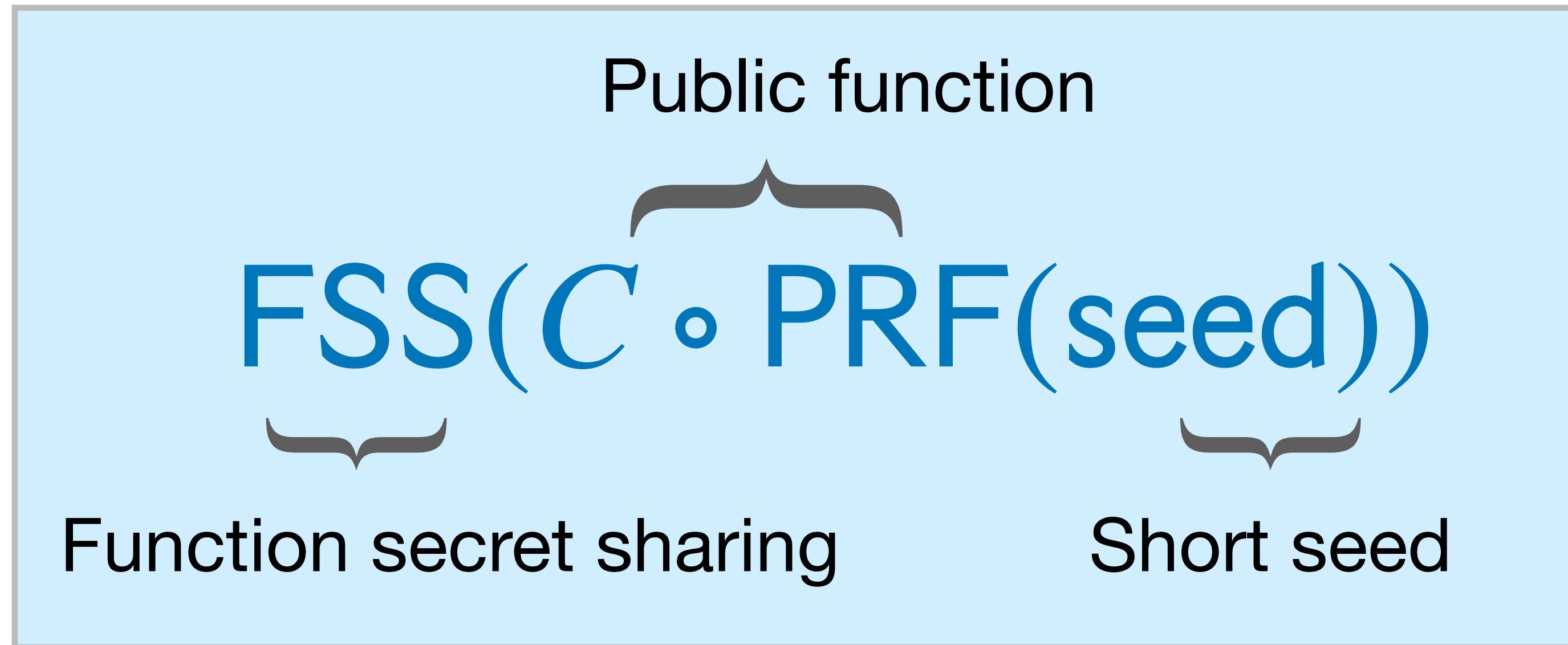
Using a PRG enables a *one-time* generation of a *fixed* amount of correlations

Back to the PCG Template Again



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

Back to the PCG Template Again

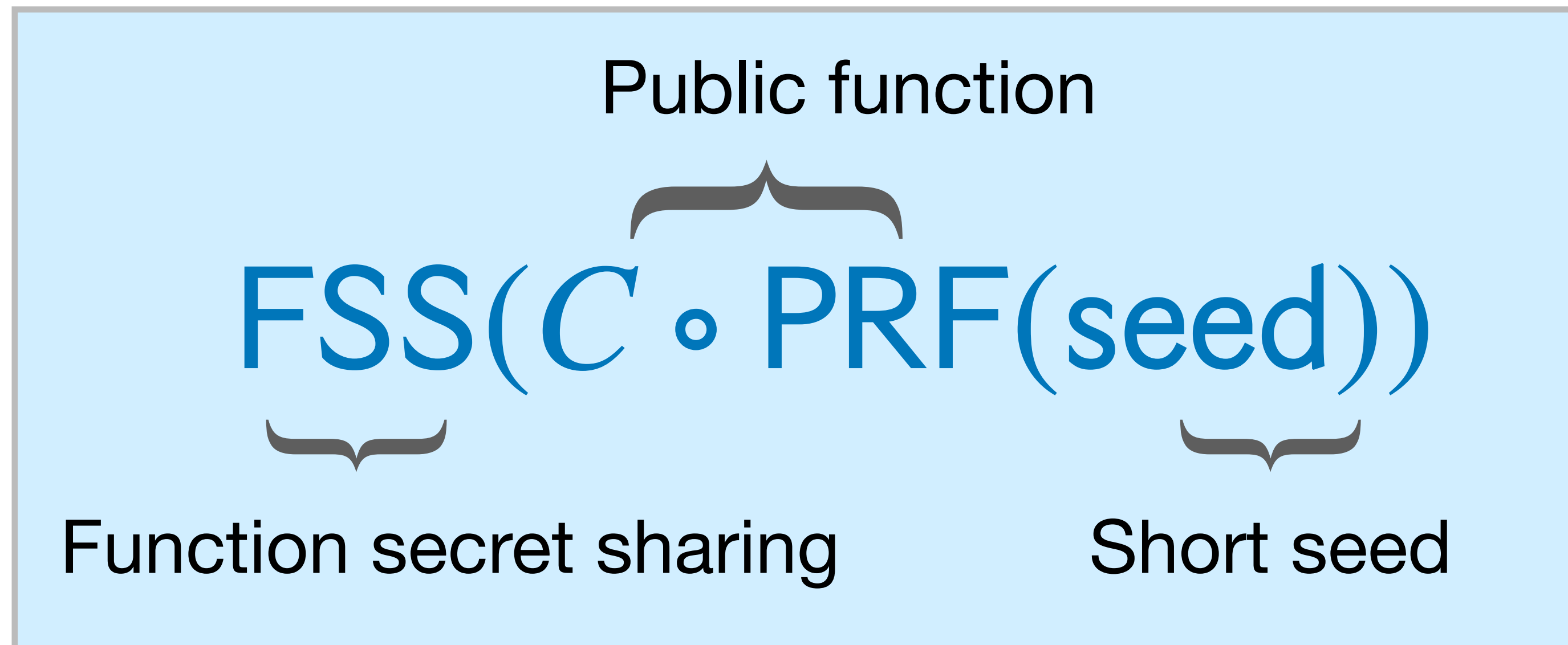


A pseudorandom correlation *function* is to a PCG what a PRF is to a PRG



Are there any FSS-friendly PRFs?

Back to the PCG Template Again



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

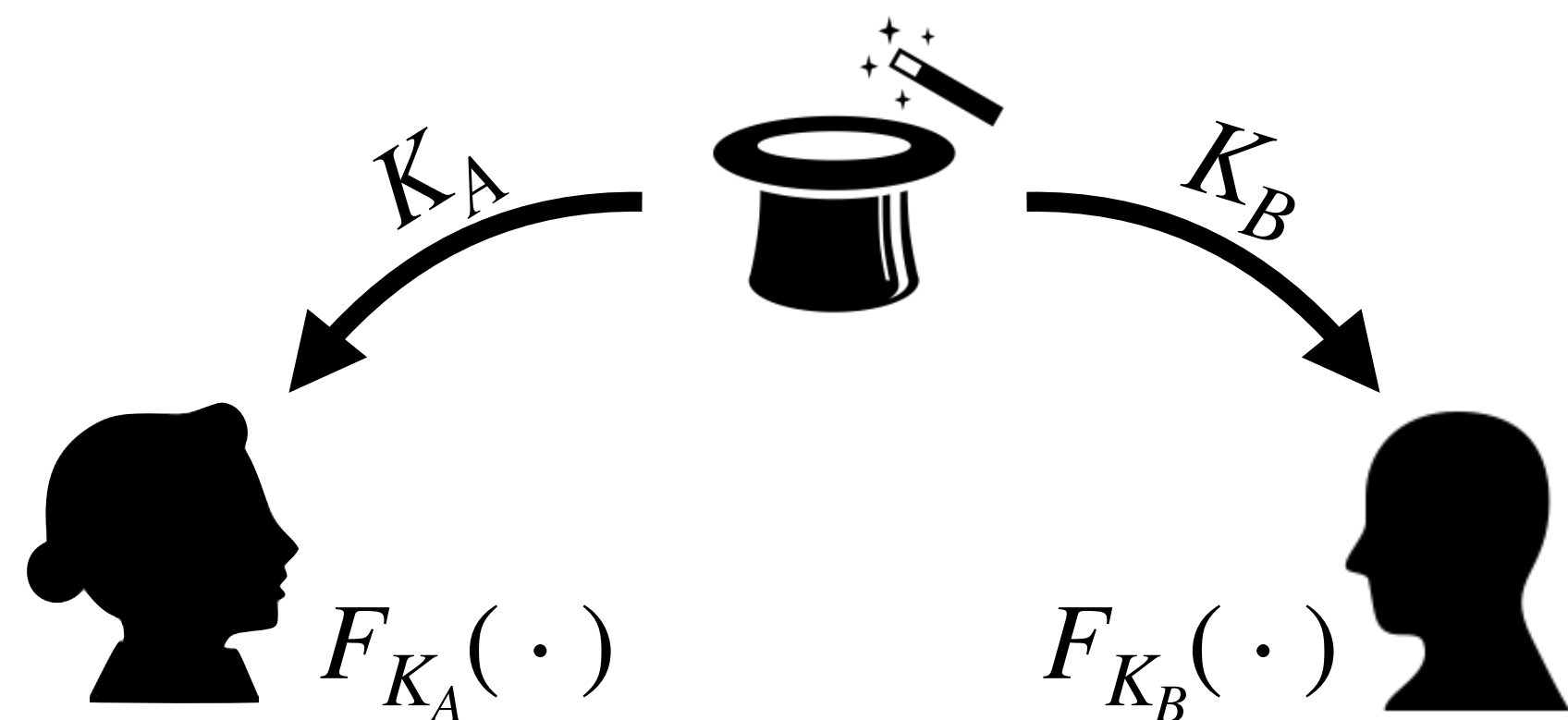
The existence of PRFs in low complexity classes yields strong limitations for learning theory



Are there any FSS-friendly PRFs?

FOCS:BCGIKS20 and Crypto:BCGIKRS22 give plausible candidates

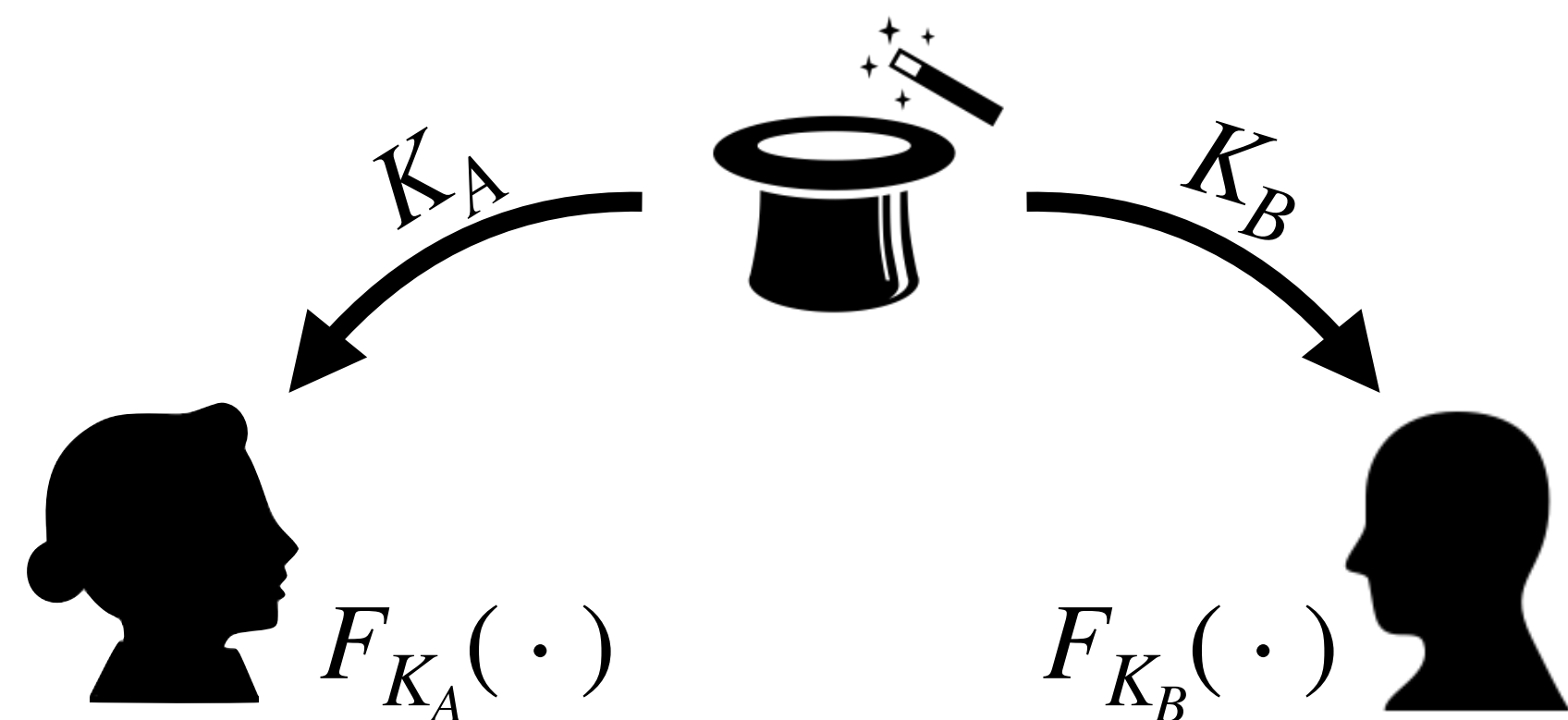
Pseudorandom Correlation Functions



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.

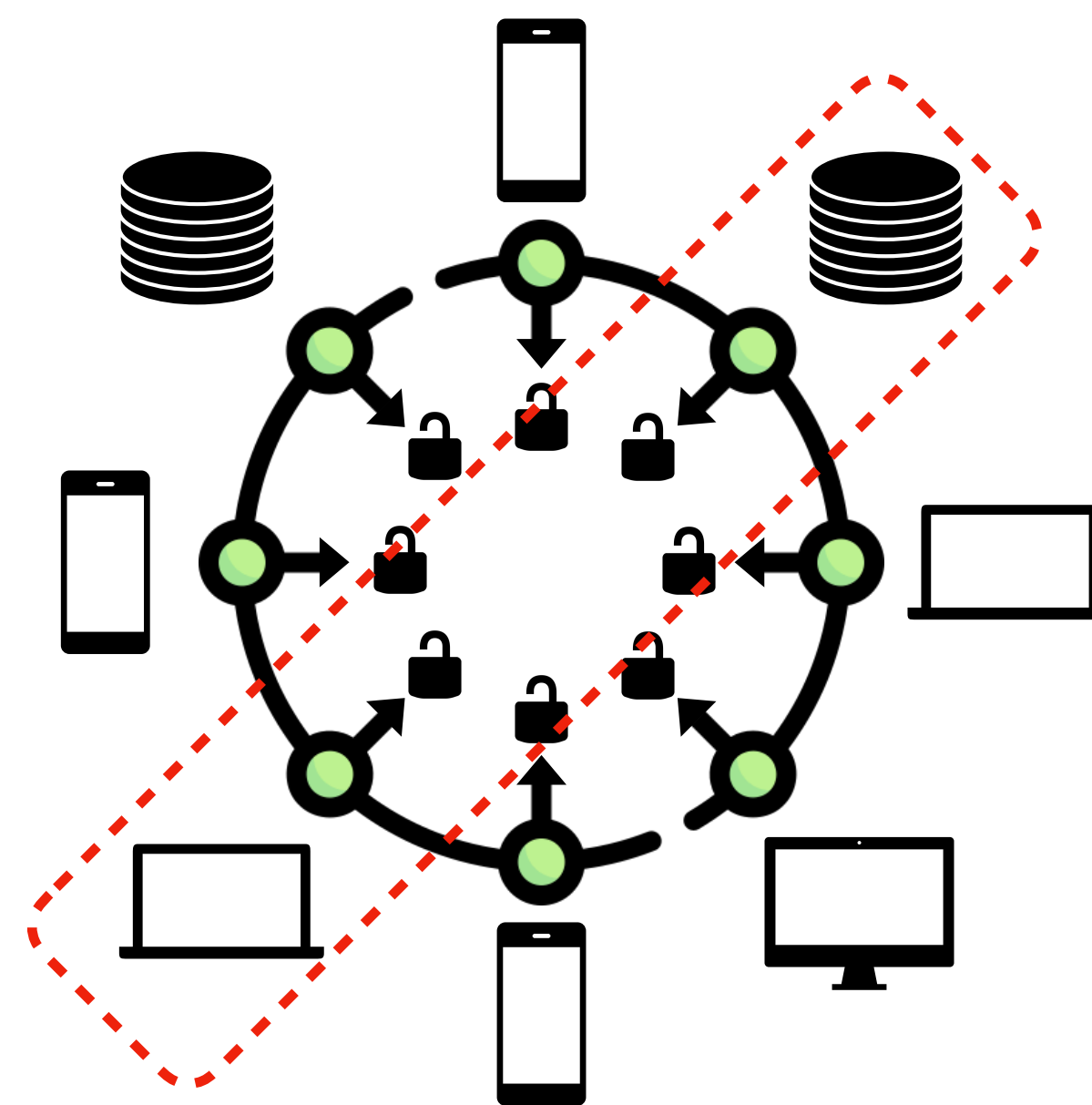
Public-Key Pseudorandom Correlation Functions



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:

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

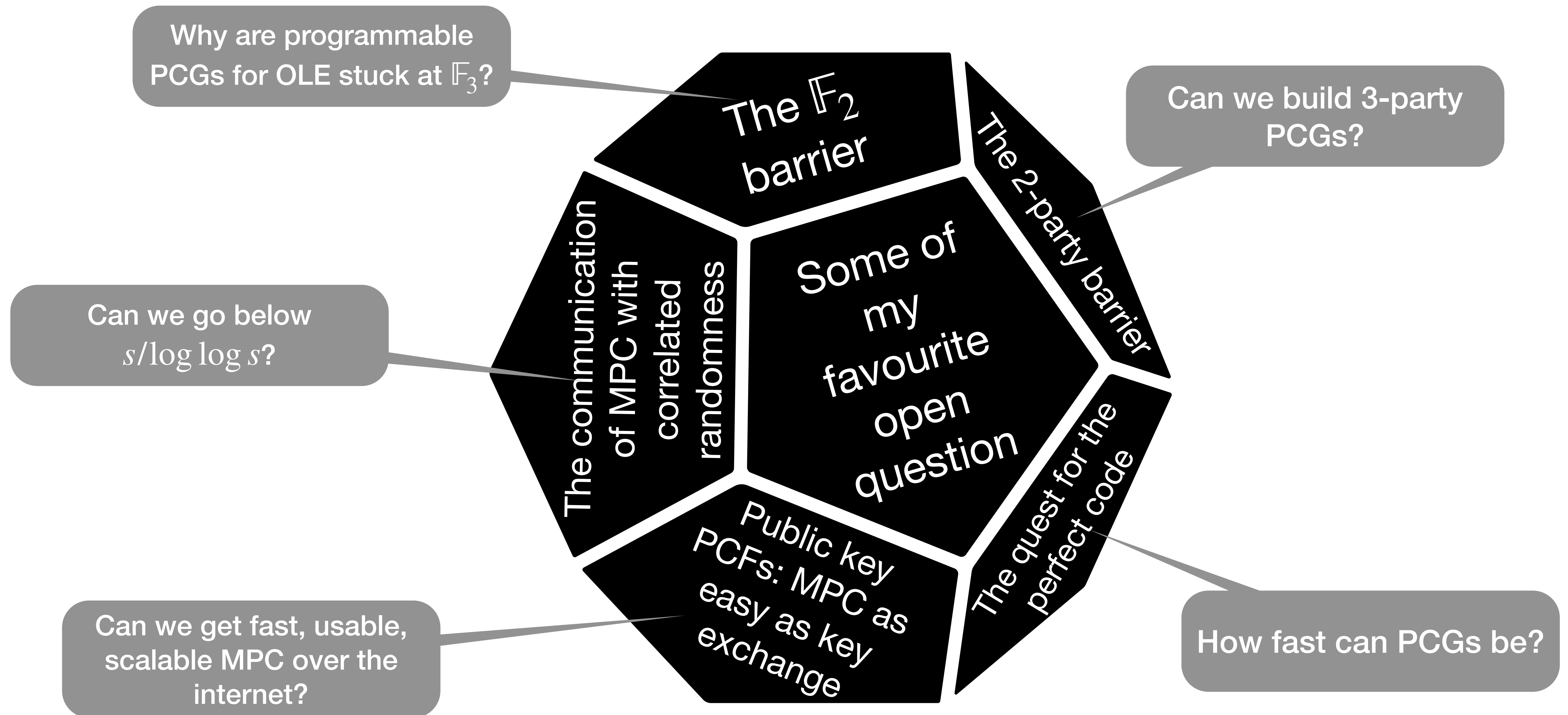
Public-Key Pseudorandom Correlation Functions

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.

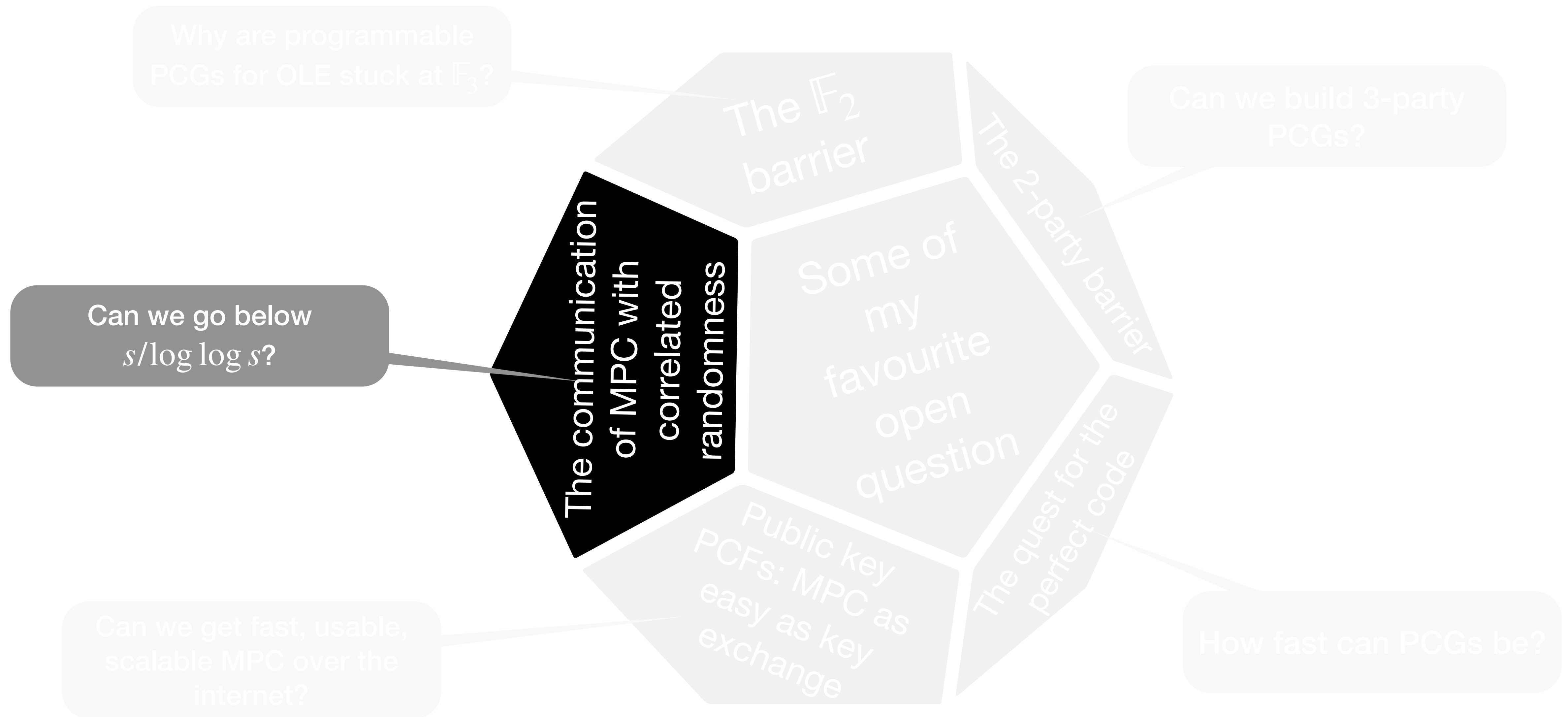
Building efficient public-key PCF is **essentially a wide-open question**: the recent work of **EC:Orlandi-Scholl-Yakubov'21** gets it for OT from QR, but efficiency is quite bad.

(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 Questions



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?

