Obfuscation

Lecture 26

Obfuscation

Obfuscation

- The art & science of making programs “unintelligible”

Obfuscation

- The art & science of making programs “unintelligible”

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
            _-_-_-_
       _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
        _-_-_-_-_-_-_-_
            _-_-_-_
}
```

from International Obfuscated C Code Contest 1988 (via Wikipedia)

Obfuscation

- The art & science of making programs "unintelligible"

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
            _-_-_-_
       _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
        _-_-_-_-_-_-_-_
            _-_-_-_
}
```

from International Obfuscated C Code Contest 1988 (via Wikipedia)

- The program should be fully functional

Obfuscation

- The art & science of making programs “unintelligible”

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
            _-_-_-_
       _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
        _-_-_-_-_-_-_-_
            _-_-_-_
}
```

from International Obfuscated C Code Contest 1988 (via Wikipedia)

  - The program should be fully functional
  - It may contain secrets that shouldn’t be revealed to the users (e.g., signature keys) — any more than executing it reveals

Obfuscation

Obfuscation

- For protecting proprietary algorithms, for crippling functionality (until license bought), for hiding potential bugs, for reducing the need for interaction with a trusted server (say for auditing purposes), ...

Obfuscation

- For protecting proprietary algorithms, for crippling functionality (until license bought), for hiding potential bugs, for reducing the need for interaction with a trusted server (say for auditing purposes), ...
- Several heuristic approaches to obfuscation exist

Obfuscation

- For protecting proprietary algorithms, for crippling functionality (until license bought), for hiding potential bugs, for reducing the need for interaction with a trusted server (say for auditing purposes), ...
- Several heuristic approaches to obfuscation exist
  - All break down against serious program analysis

Cryptographic Obfuscation

Cryptographic Obfuscation

- Obfuscation using cryptography?

Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion

Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions

Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions
- Cryptography using obfuscation

Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions
- Cryptography using obfuscation
  - If realized, obfuscation can be used to instantiate various other powerful cryptographic primitives

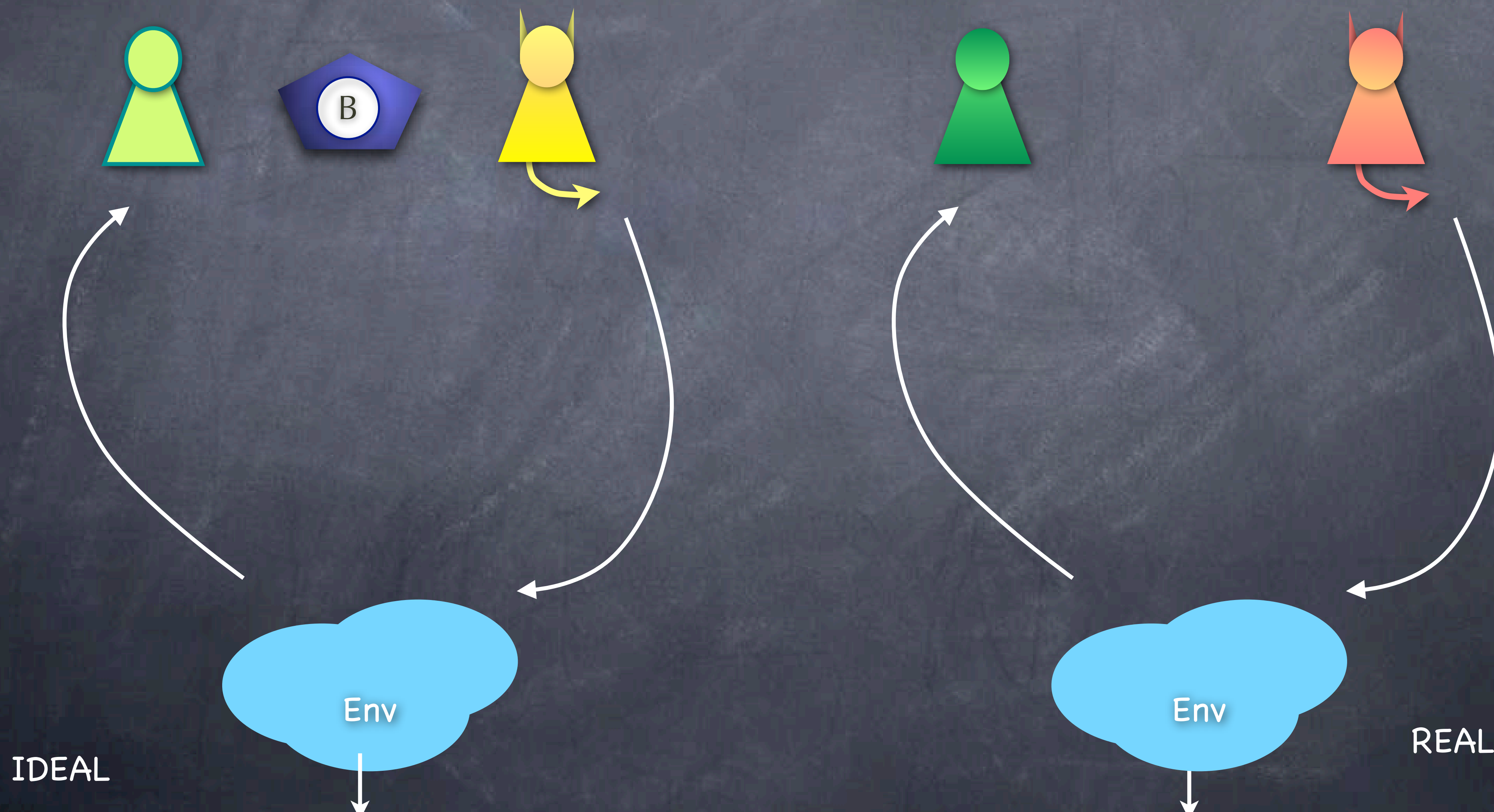
Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions
- Cryptography using obfuscation
  - If realized, obfuscation can be used to instantiate various other powerful cryptographic primitives
  - Toy example: PKE from SKE. Obfuscate the SKE encryption program with the key inside (and a PRF for generating randomness from the plaintext), and release as public-key

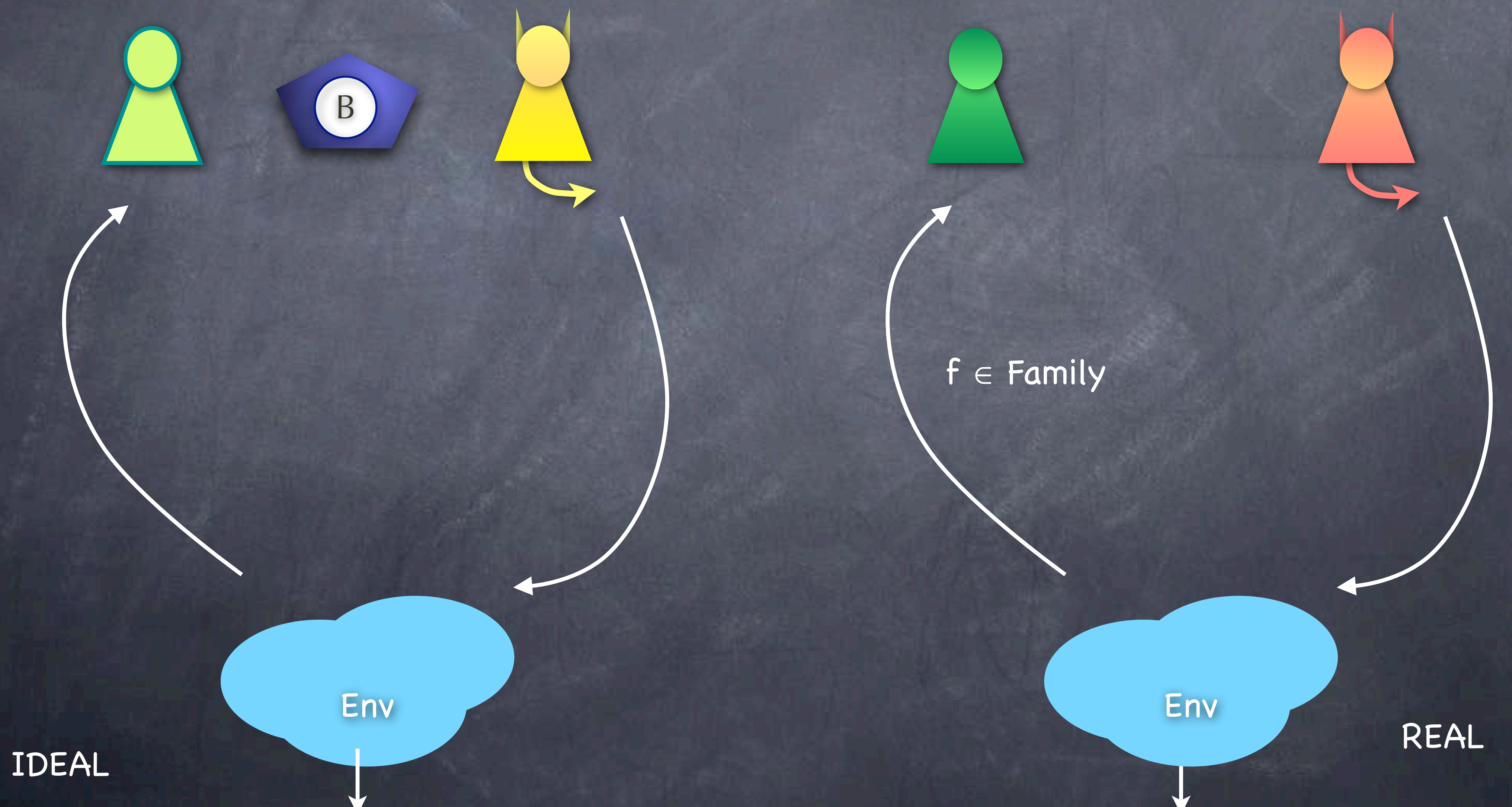
Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions
- Cryptography using obfuscation
  - If realized, obfuscation can be used to instantiate various other powerful cryptographic primitives
  - Toy example: PKE from SKE. Obfuscate the SKE encryption program with the key inside (and a PRF for generating randomness from the plaintext), and release as public-key
    - Or IBE: Encryption also MACs (ID,ciphertext). Decryption key for ID is a program that checks ID/MAC before decrypting

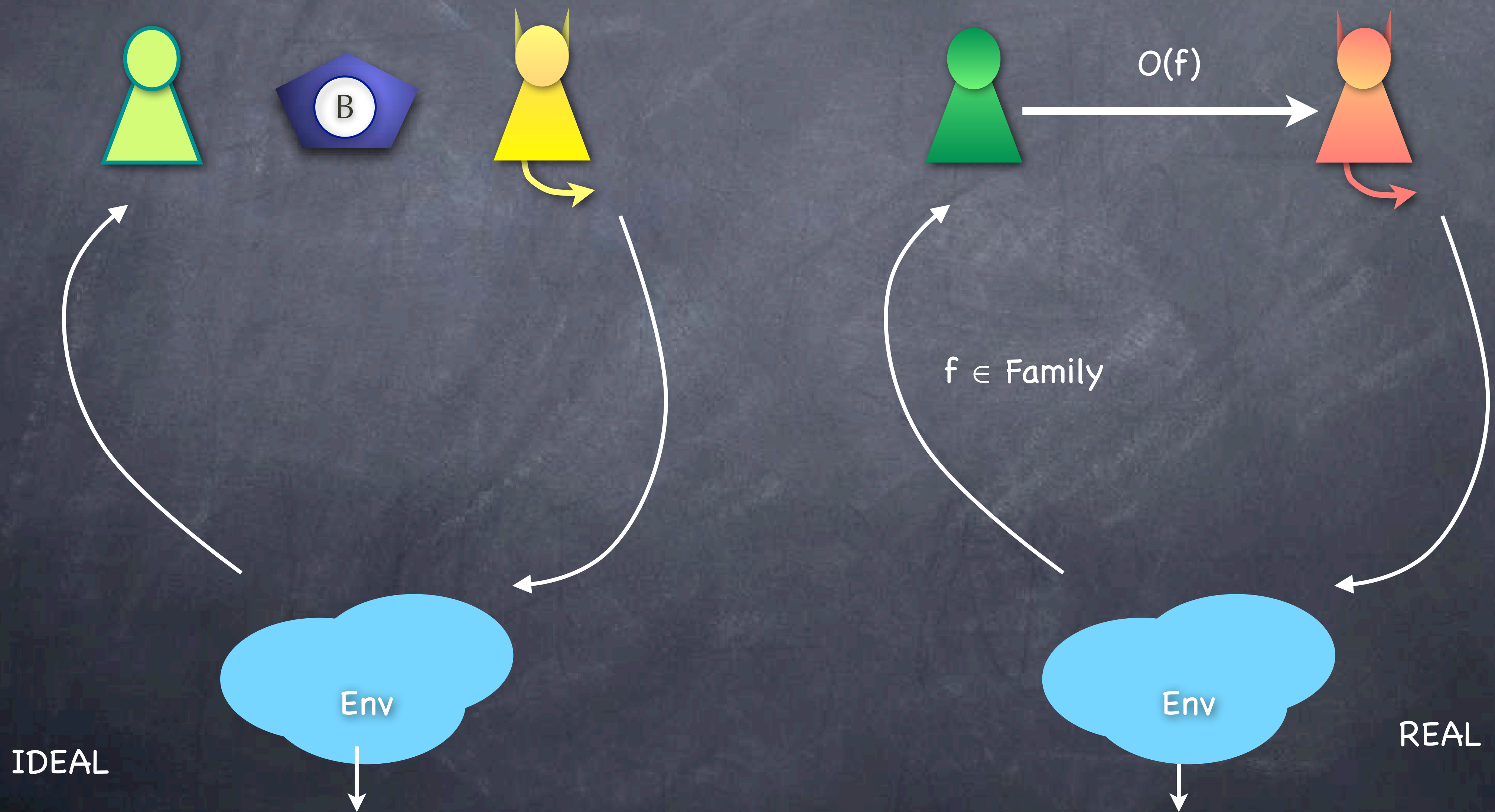
Defining Obfuscation: First Try



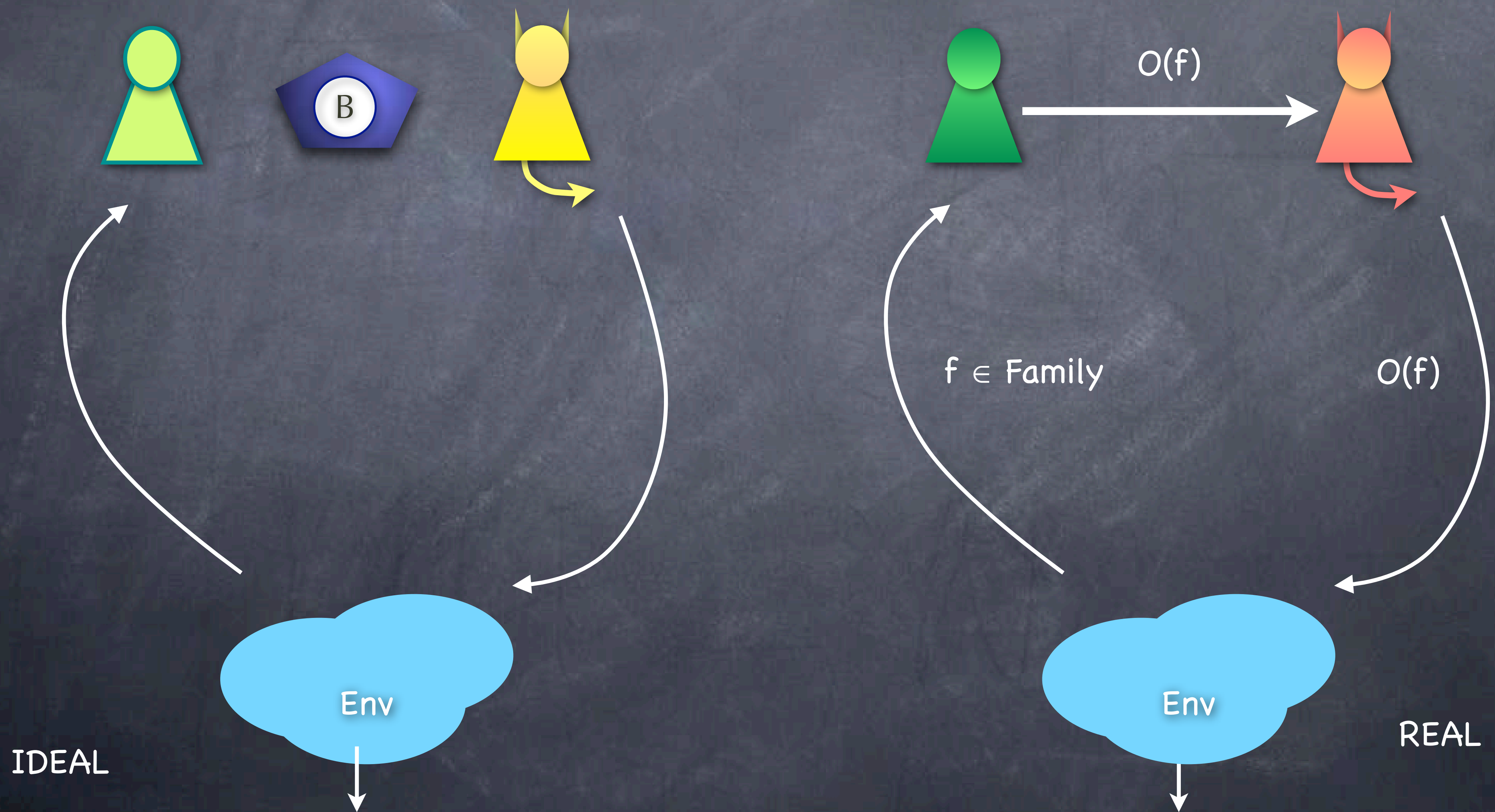
Defining Obfuscation: First Try



Defining Obfuscation: First Try

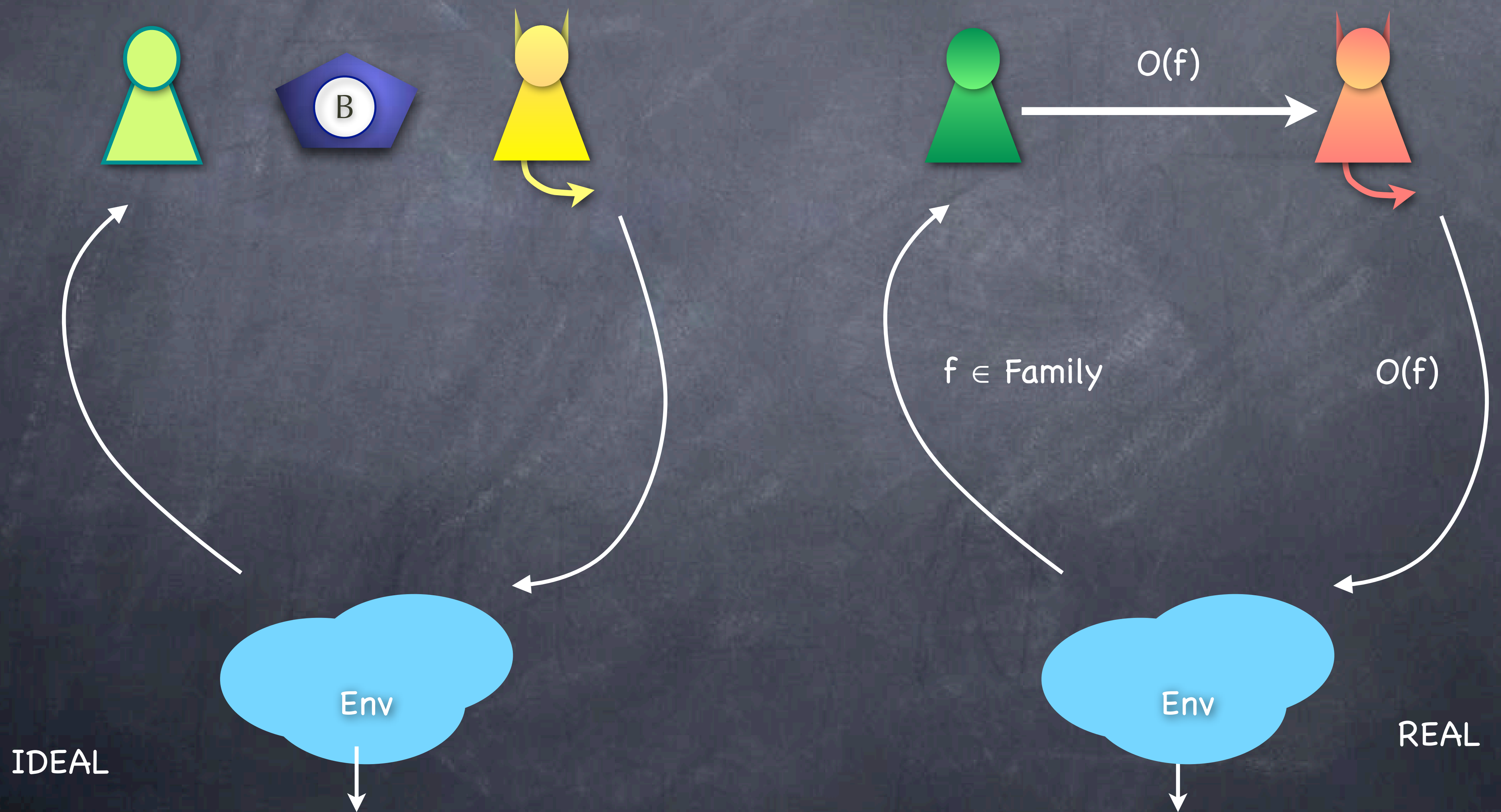


Defining Obfuscation: First Try



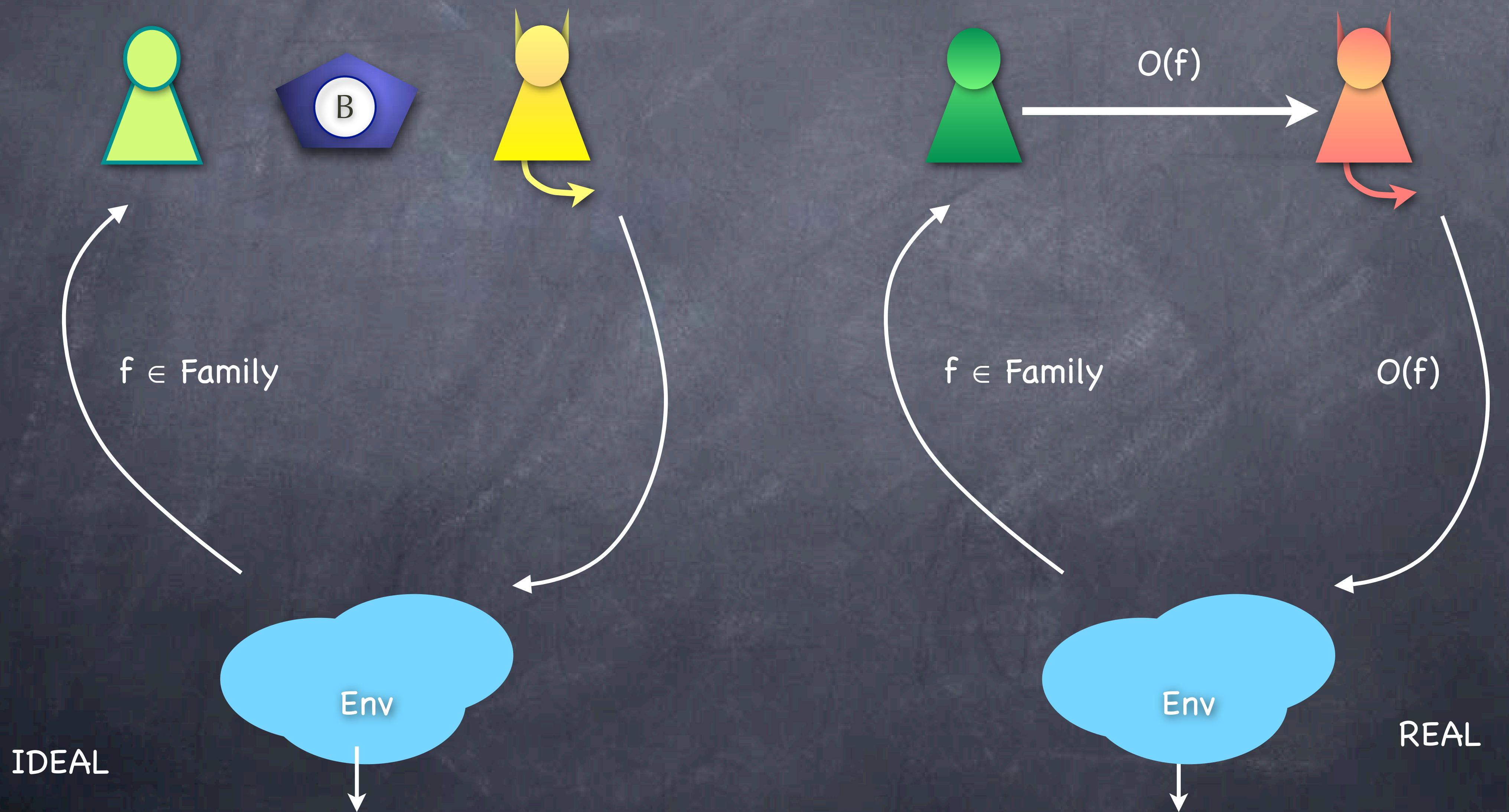
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



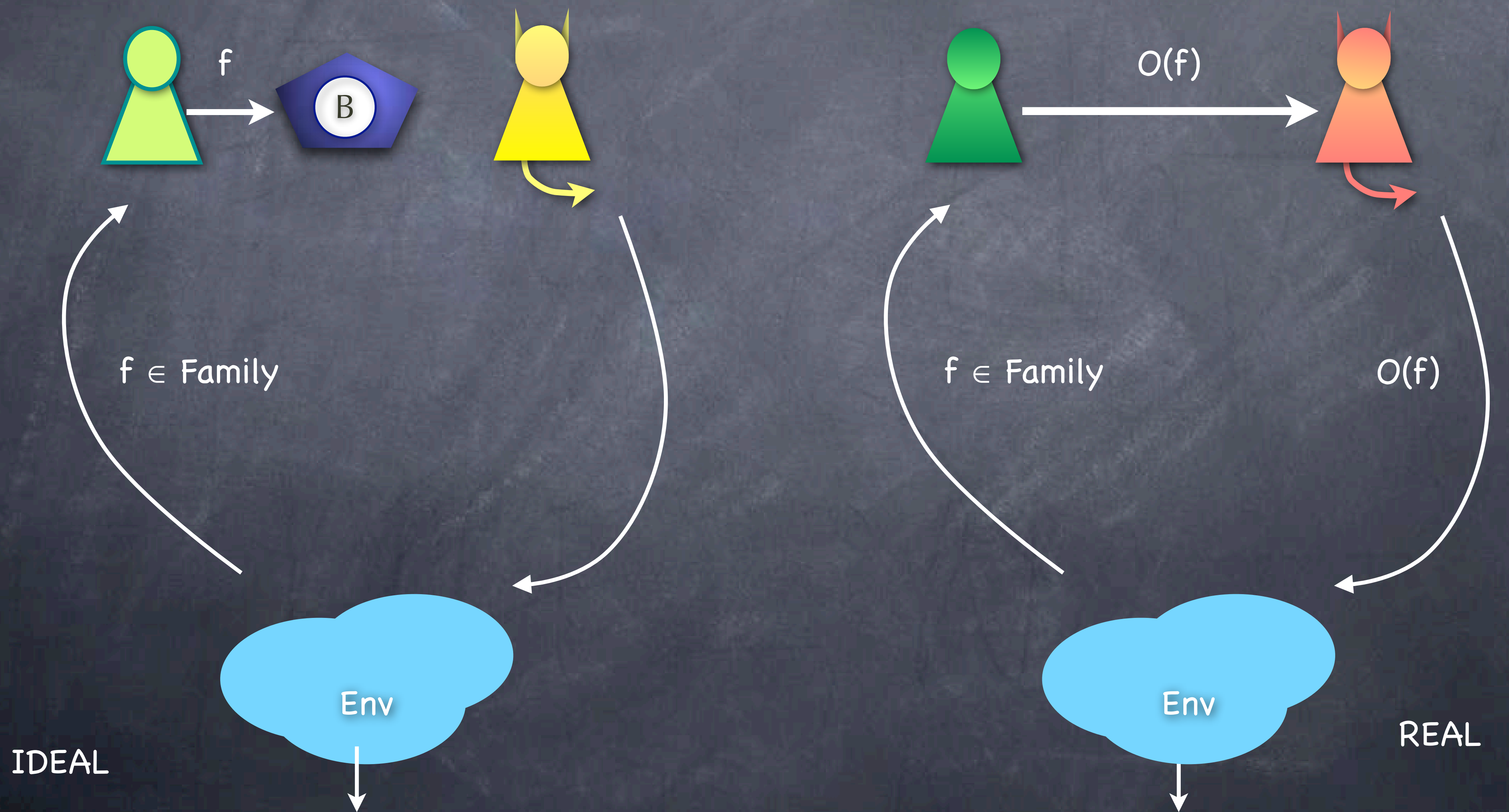
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



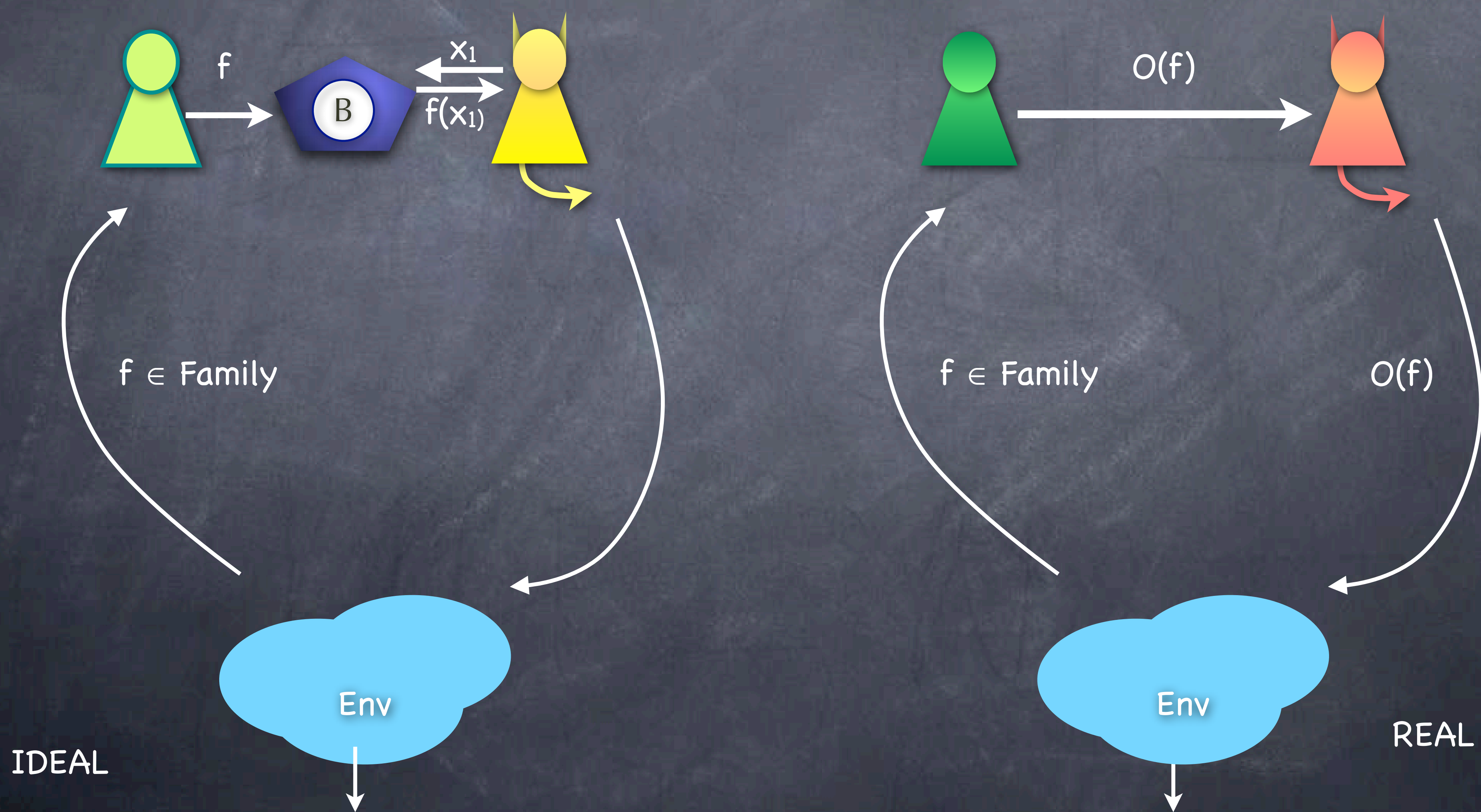
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



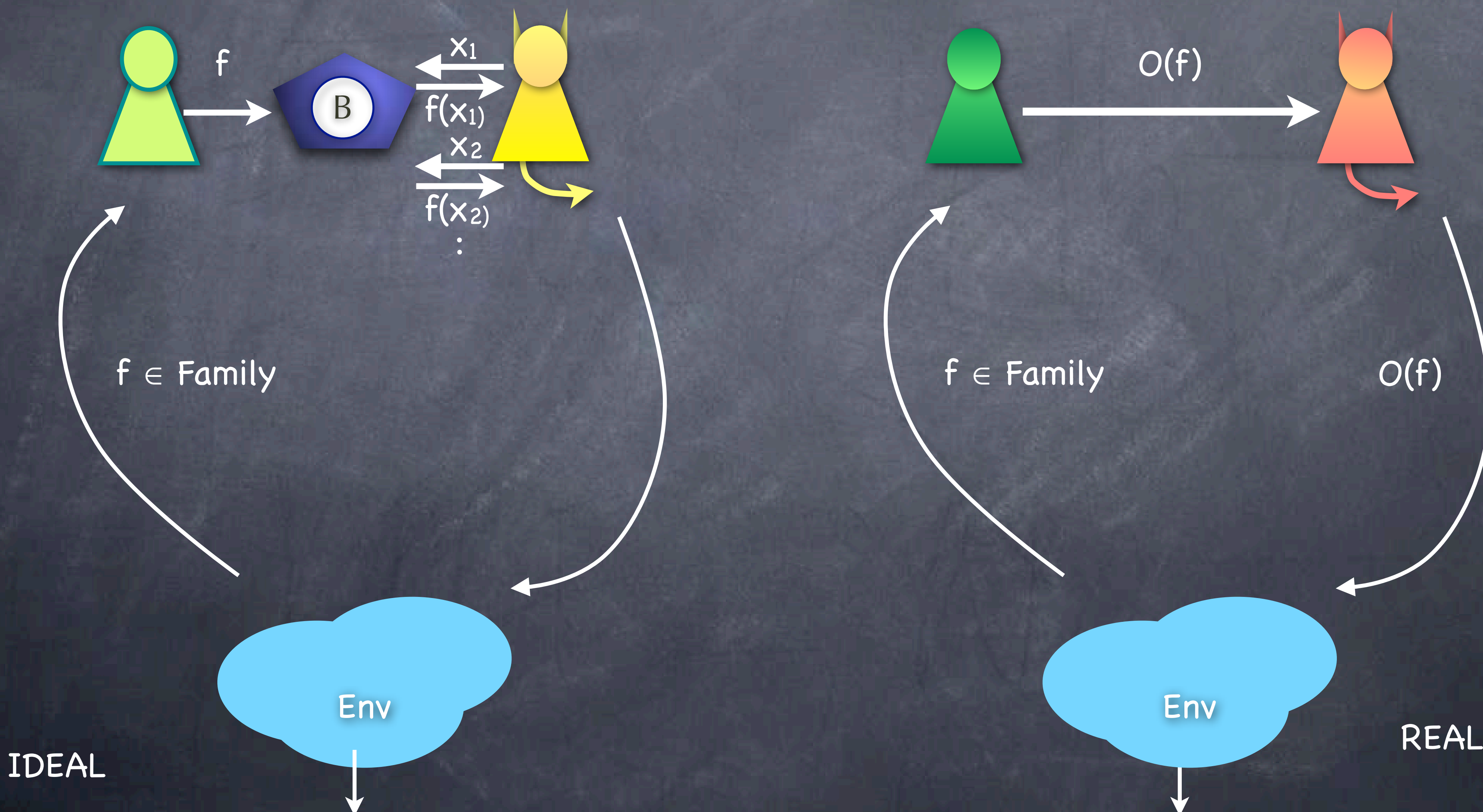
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



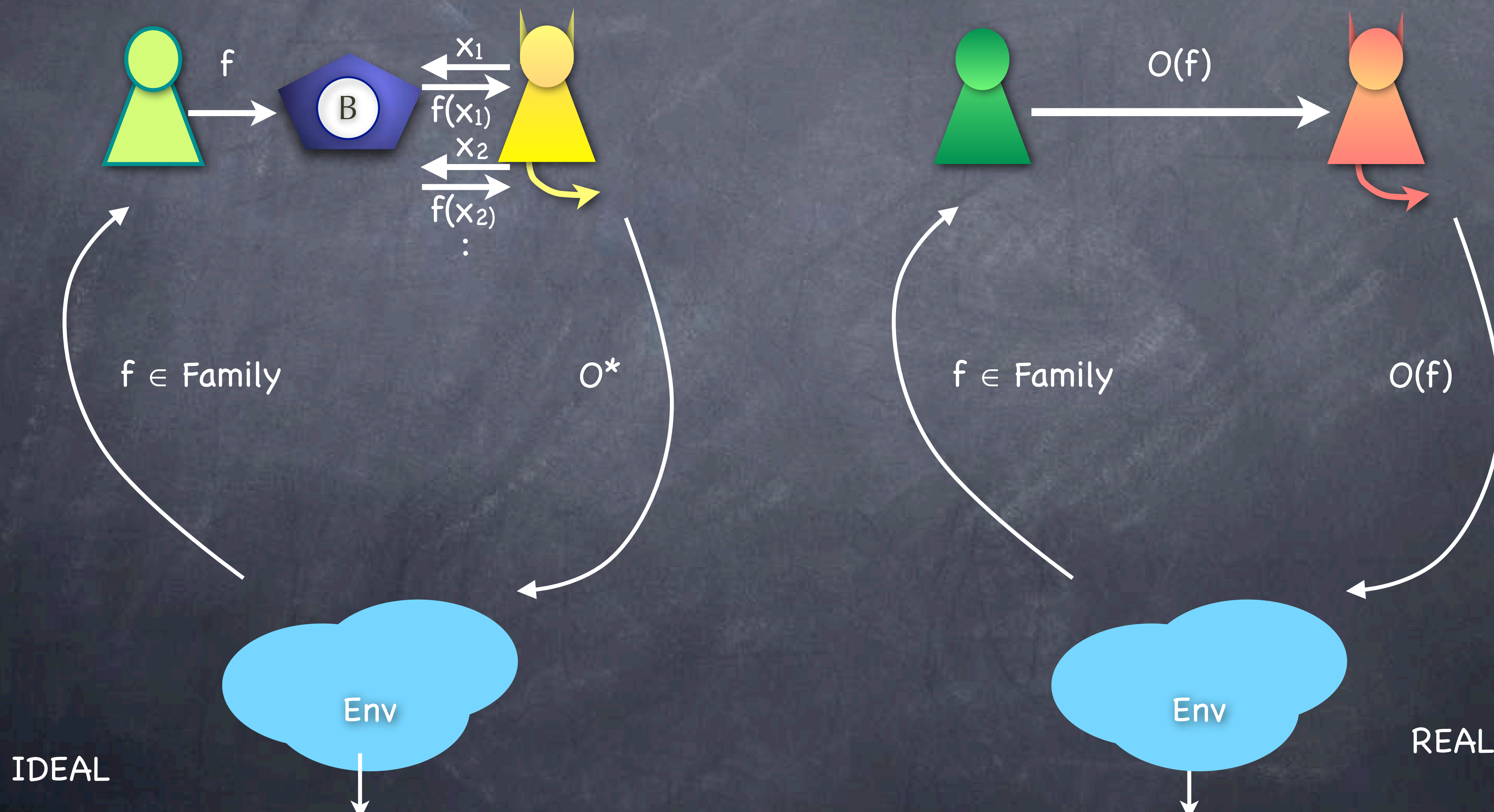
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



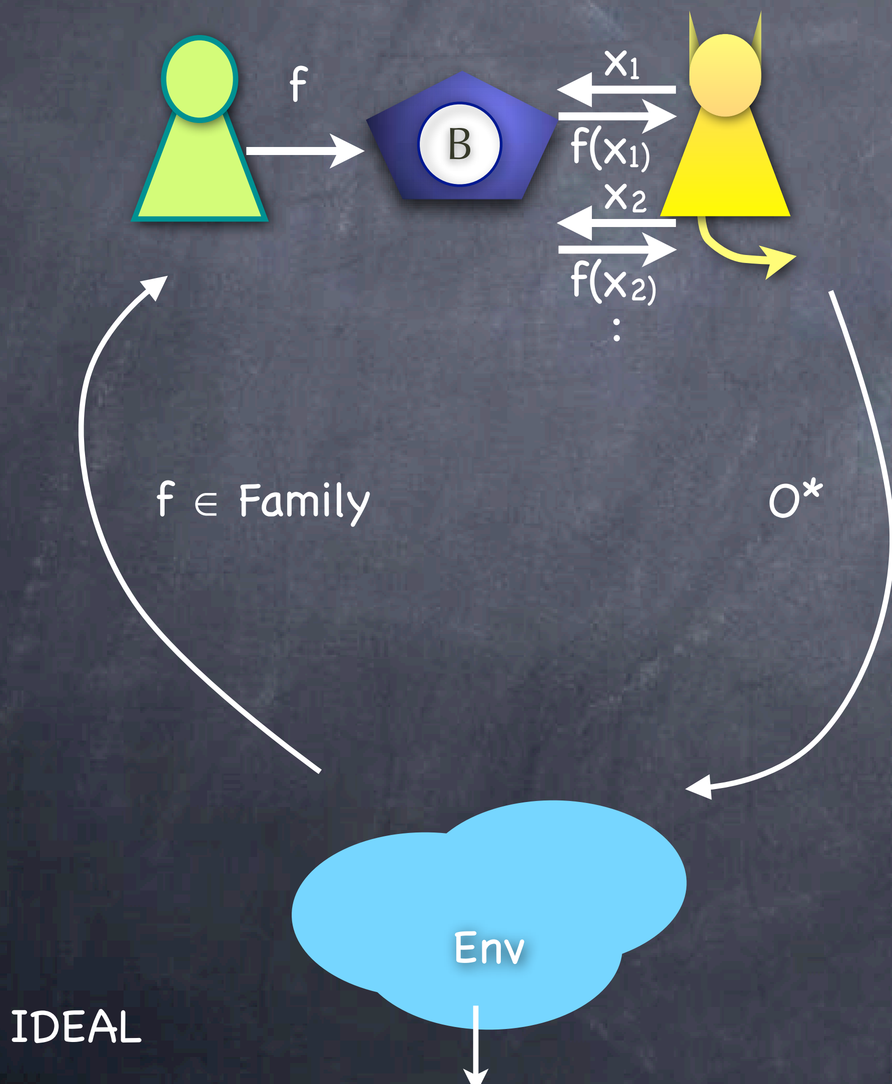
Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Secure (and correct) if:

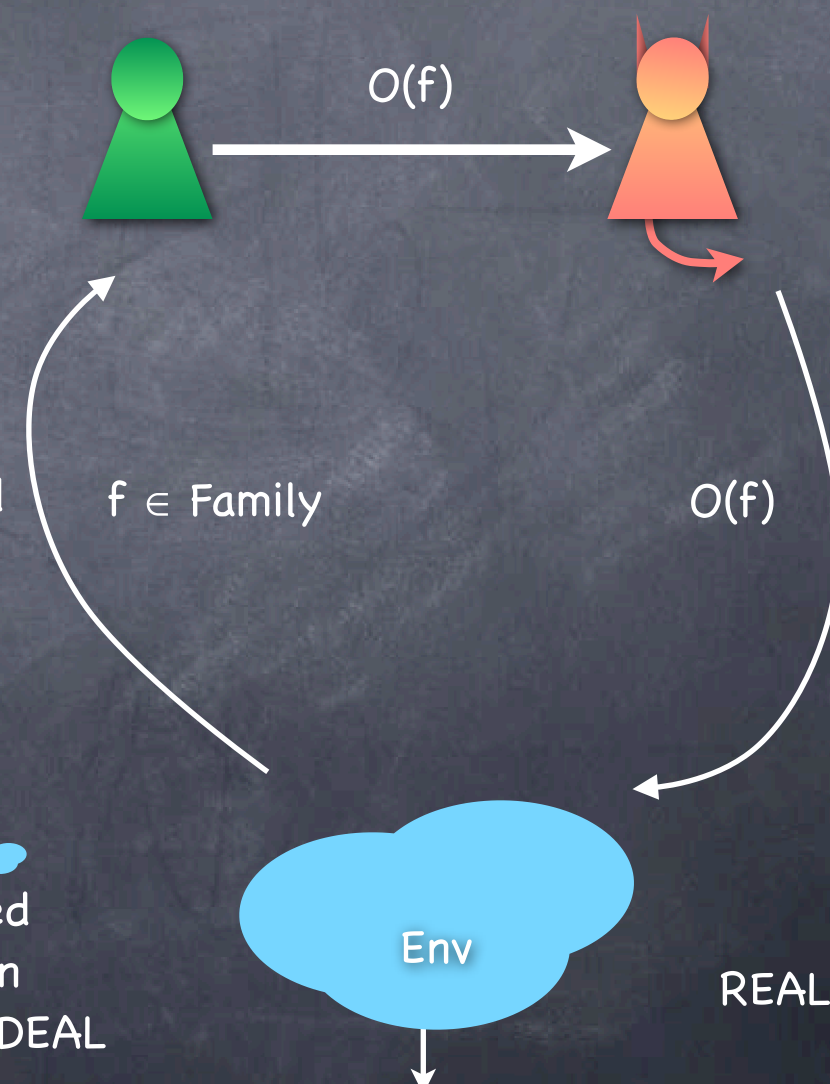
∀

∃ s.t.

∀

output of

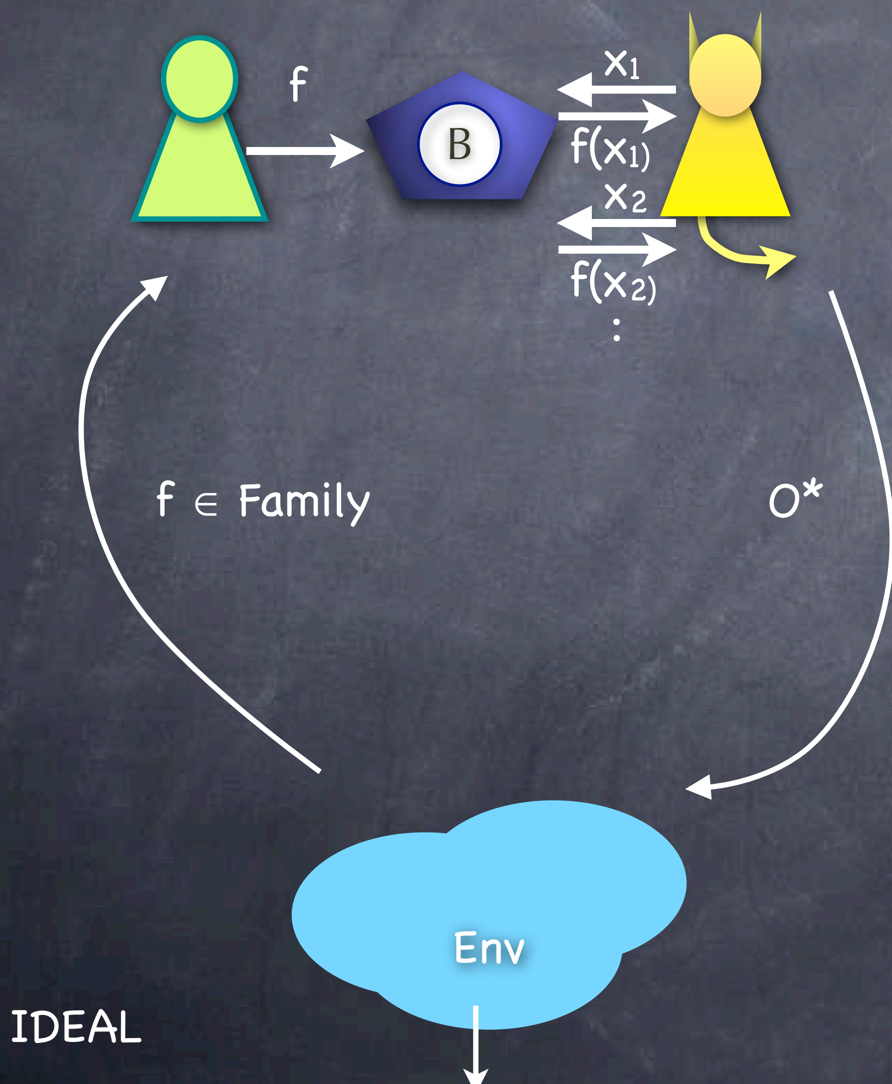
is distributed identically in REAL and IDEAL



Defining Obfuscation: First Try

Note: Considers only corrupt receiver

Too strong! Requires family to be <u>learnable</u> from black-box access



Secure (and correct) if:

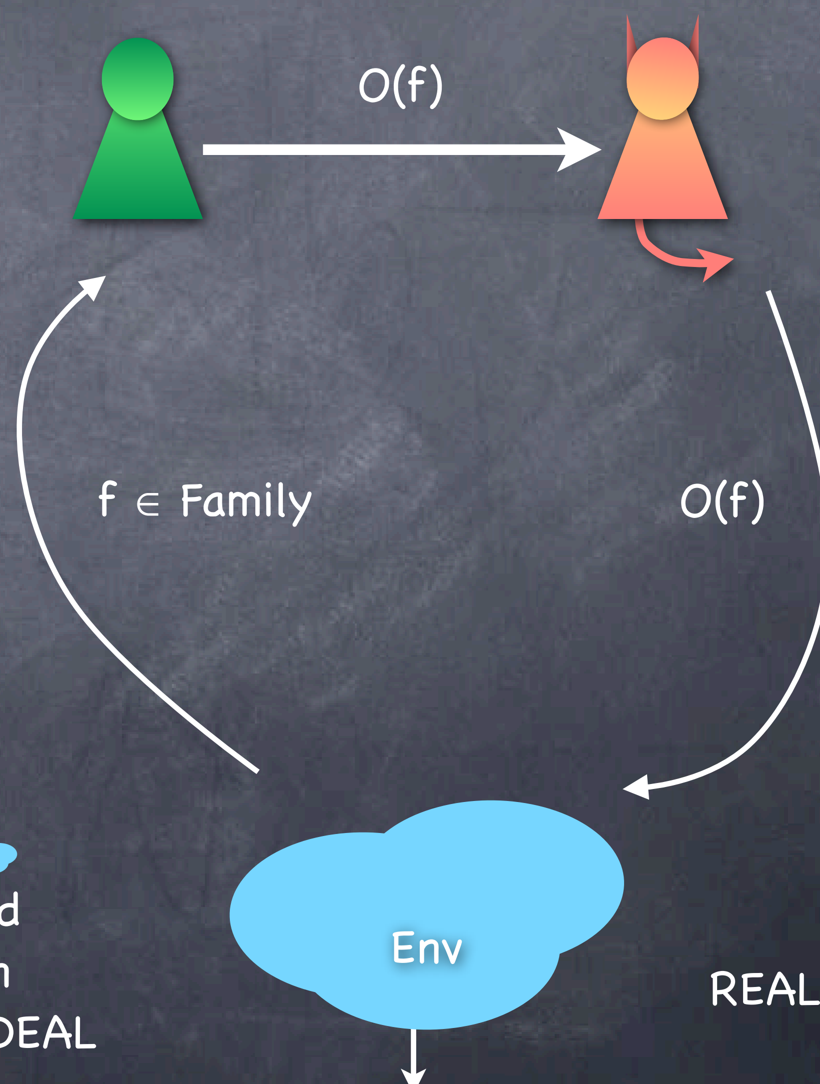
∀

∃ s.t.

∀

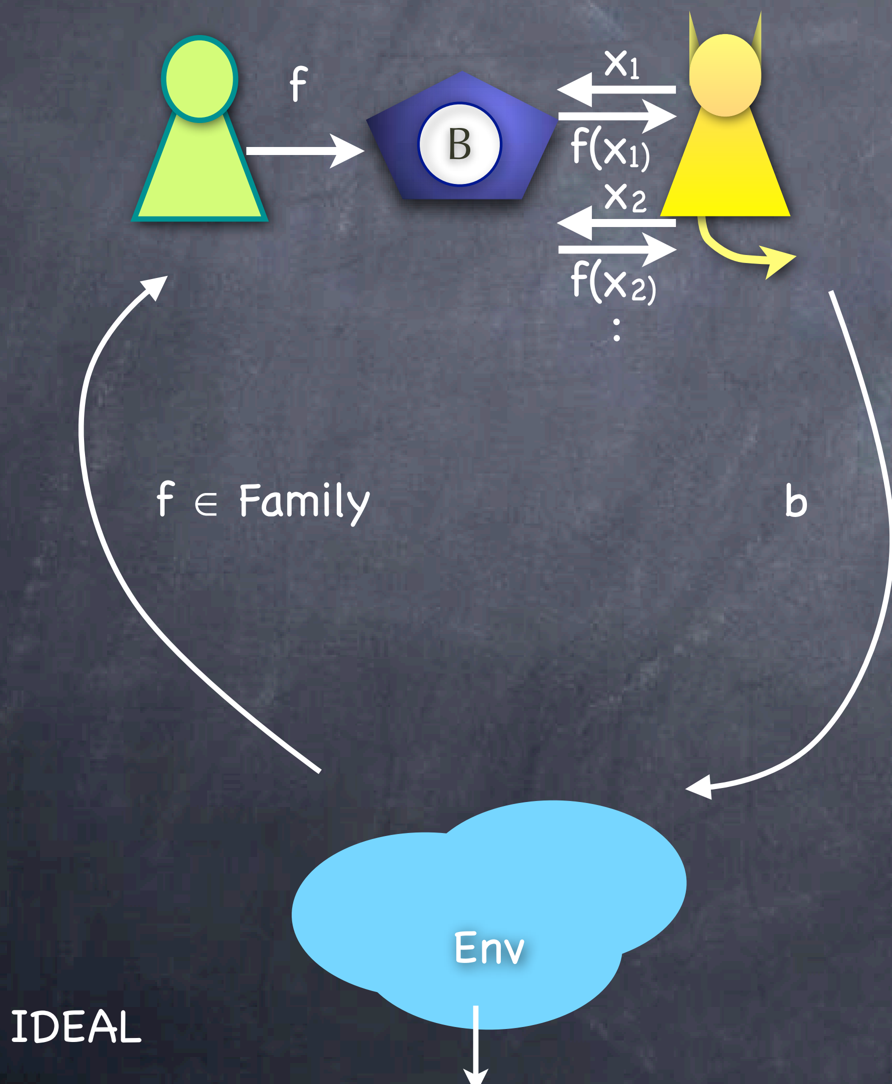
output of

is distributed identically in REAL and IDEAL



Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Secure (and correct) if:

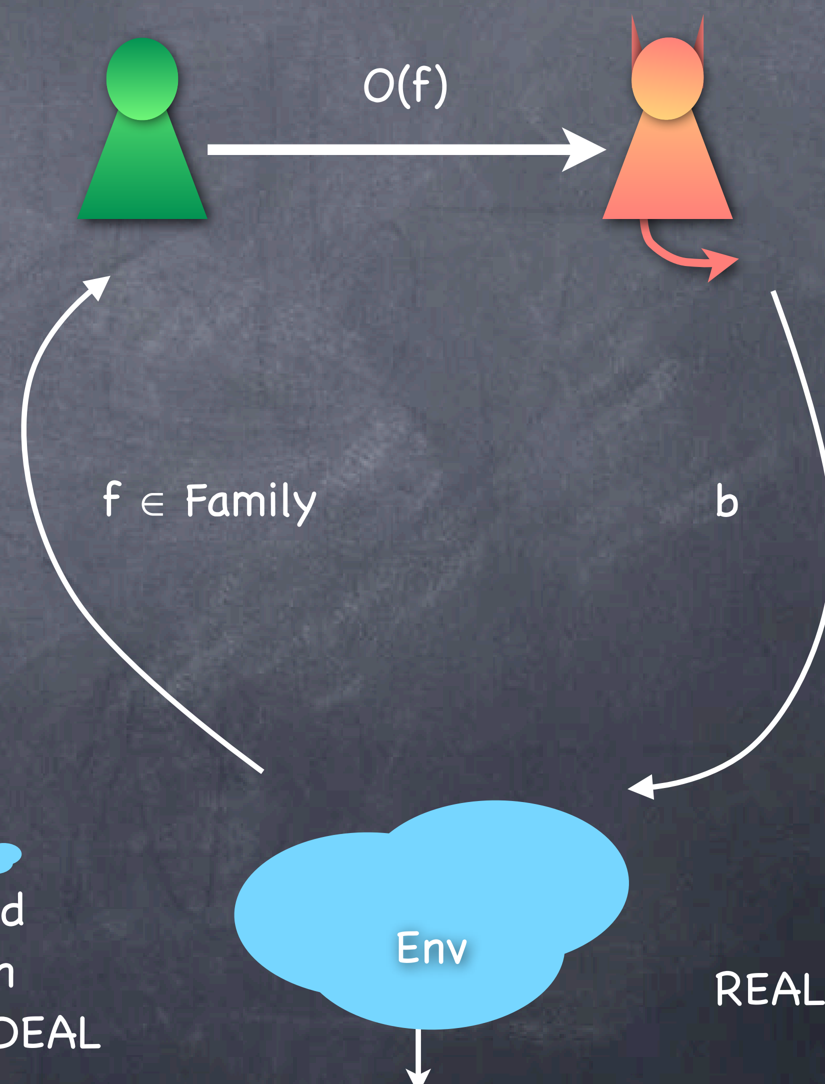
∀

∃ s.t.

∀

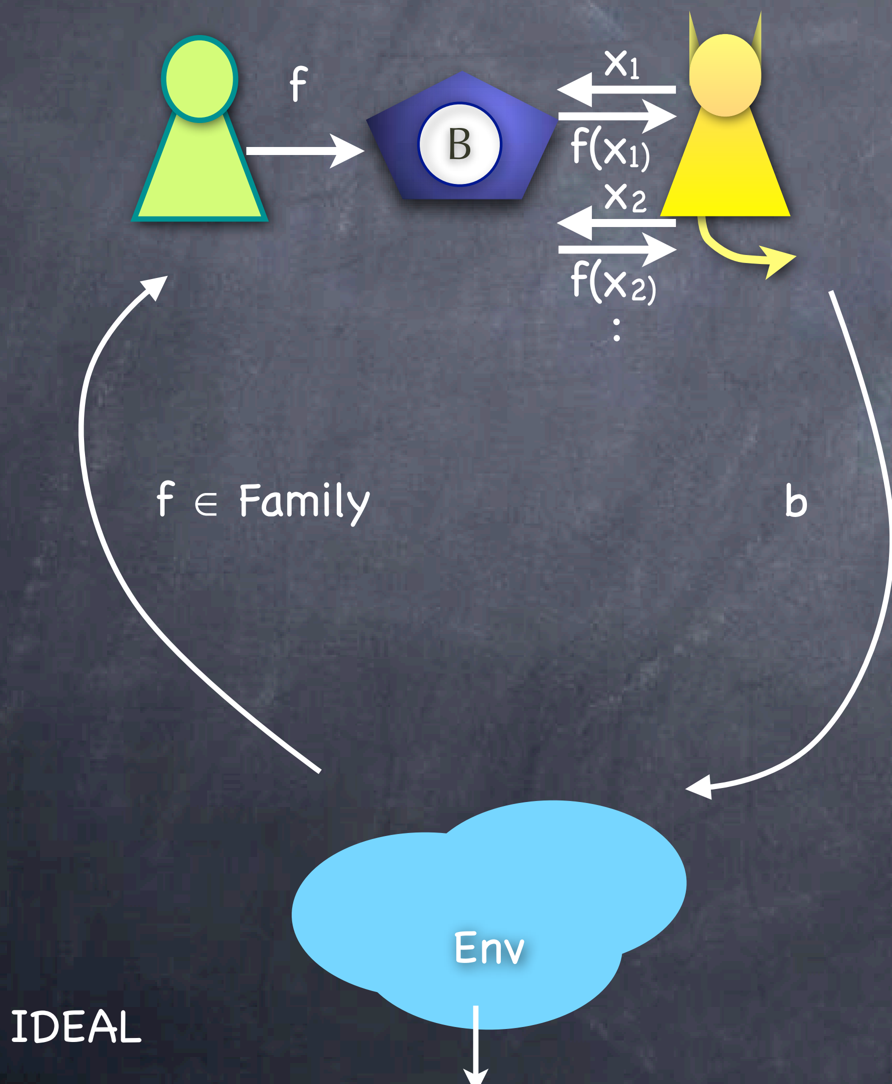
output of

is distributed identically in REAL and IDEAL



Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Secure (and correct) if:

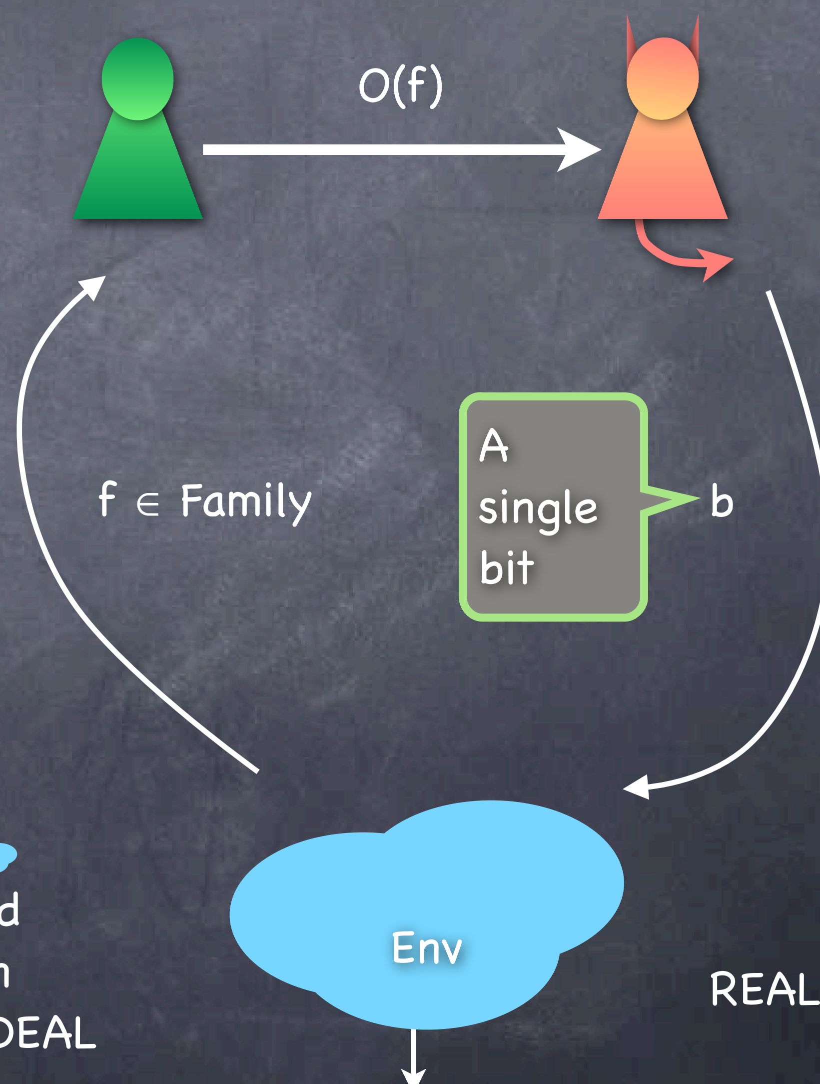
∀

∃ s.t.

∀

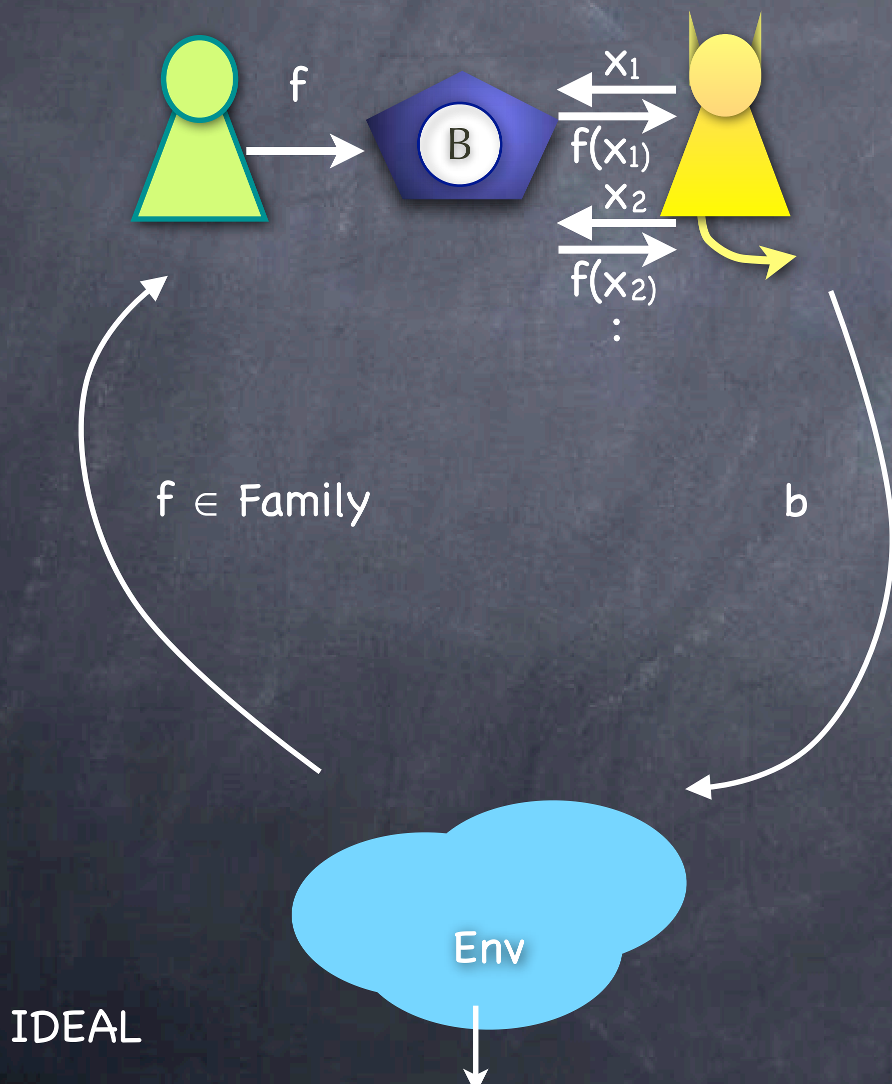
output of

is distributed identically in REAL and IDEAL



Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Virtual Black-Box (VBB) Obfuscation

Secure (and correct) if:

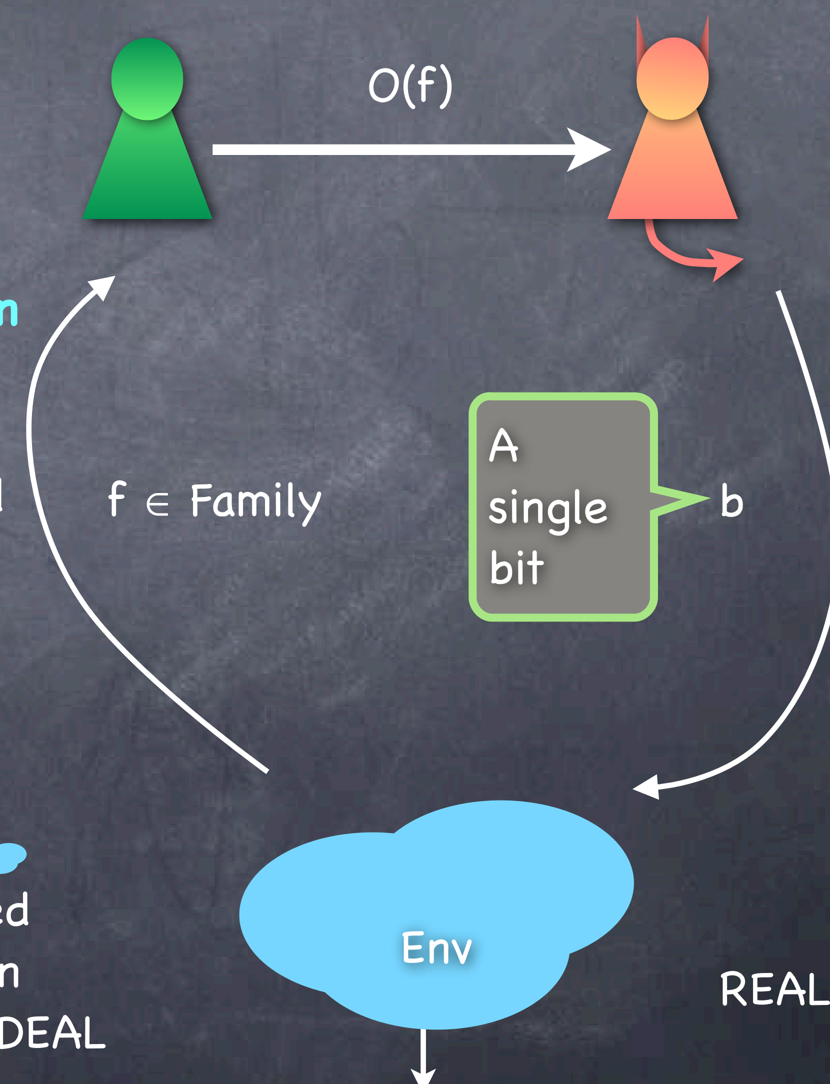
∀

∃ s.t.

∀

output of

is distributed identically in REAL and IDEAL



Impossibility of Obfuscation

Impossibility of Obfuscation

- VBB obfuscation is impossible in general

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
    - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:
    - If input is of the form $(0,\alpha)$ output β

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:
    - If input is of the form $(0,\alpha)$ output β
    - If input is of the form (1,P) for a program P, run P with input $(0,\alpha)$ and if it outputs β, output (α,β)

Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:
    - If input is of the form (0, α) output β
    - If input is of the form (1,P) for a program P, run P with input (0, α) and if it outputs β, output (α, β)
  - When $P_{\alpha,\beta}$ is run on its own code, it outputs (α, β). Can learn, e.g., first bit of α. In the ideal world, need to guess!

Possibility of Obfuscation

Possibility of Obfuscation

- For simple function families

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)
- For weaker definitions

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)
- For weaker definitions
- Hardware assisted

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)
- For weaker definitions
- Hardware assisted
- In idealized models (random oracle model, generic group model, etc.)

Possibility of Obfuscation

- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)
- For weaker definitions
- Hardware assisted
- In idealized models (random oracle model, generic group model, etc.)
  - Need a suitable representation of the function

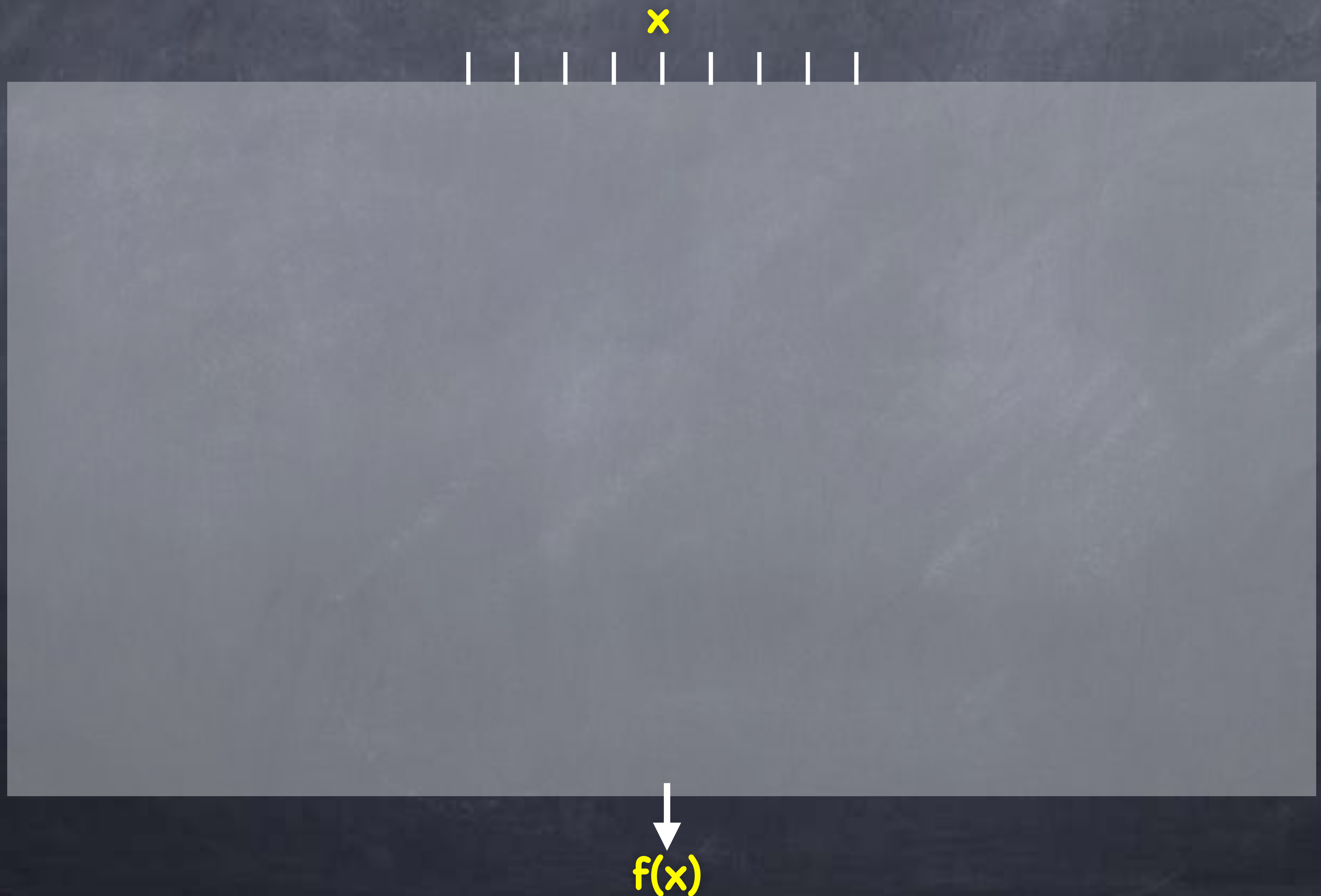
Matrix Programs

Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of $2N$ $w \times w$ matrices ($N = poly(n)$)

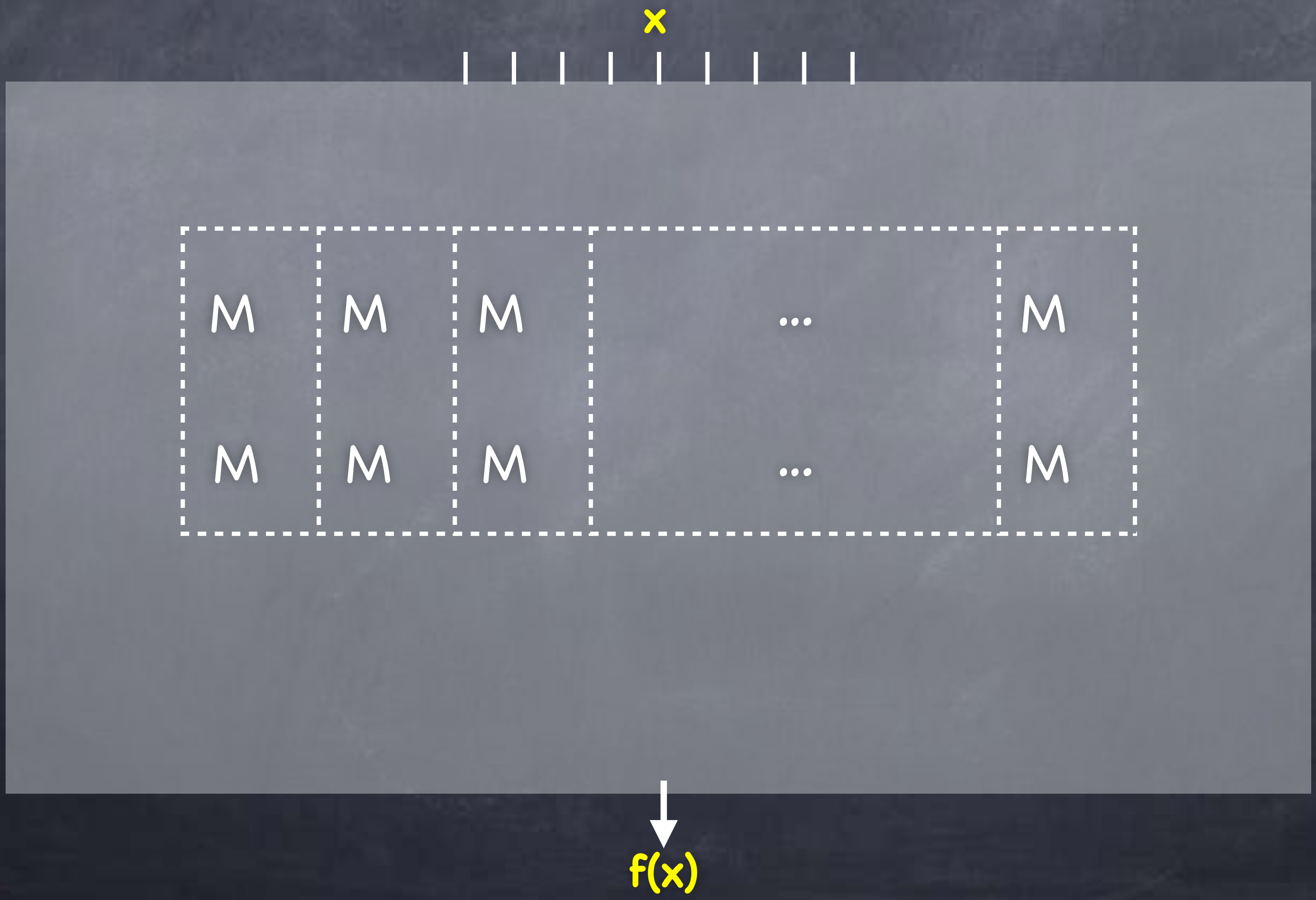
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of 2N $w \times w$ matrices (N = poly(n))



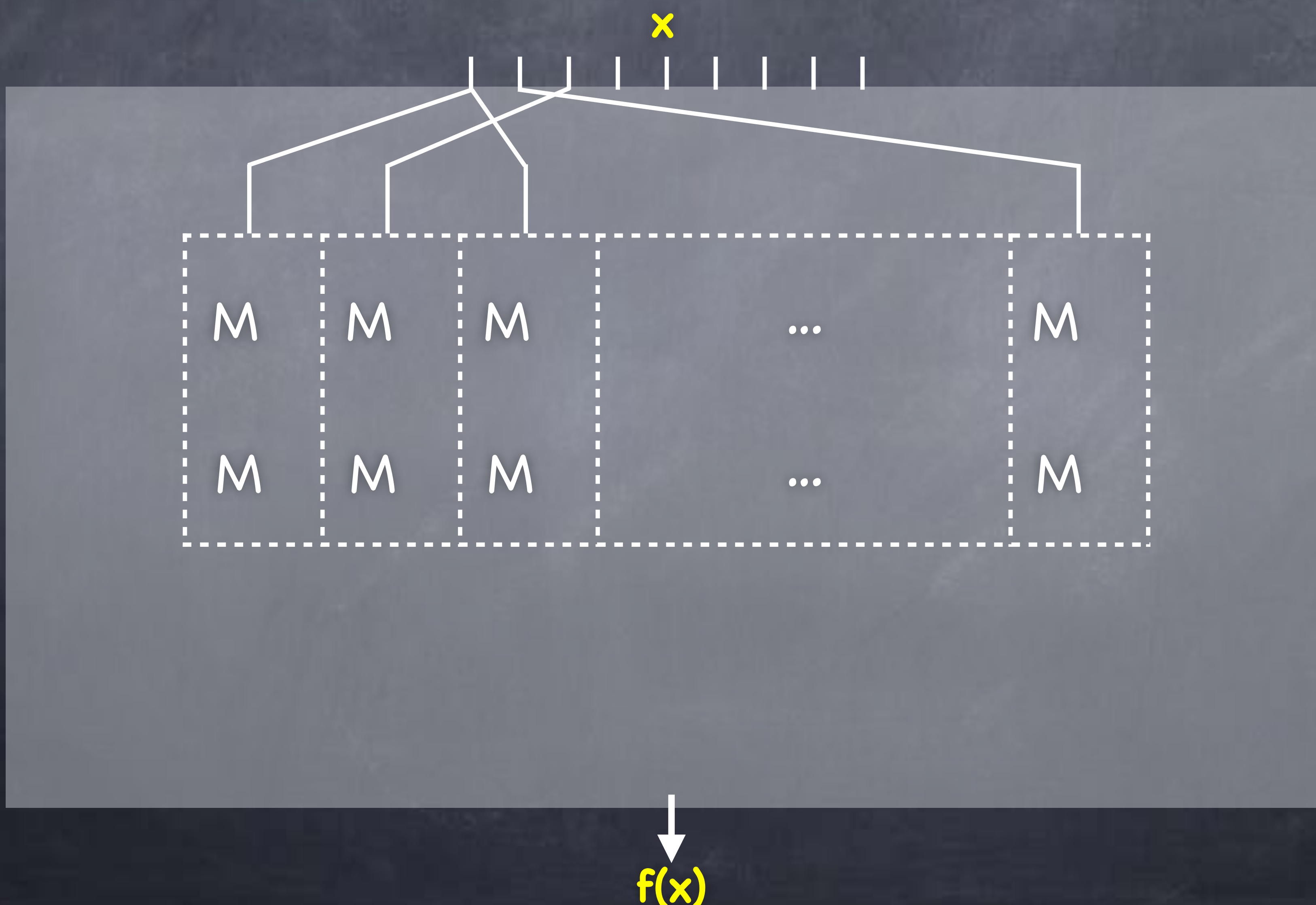
Matrix Programs

- $f : \{0,1\}^n \to \{0,1\}$ using a set of 2N $w \times w$ matrices ($N = \text{poly}(n)$)



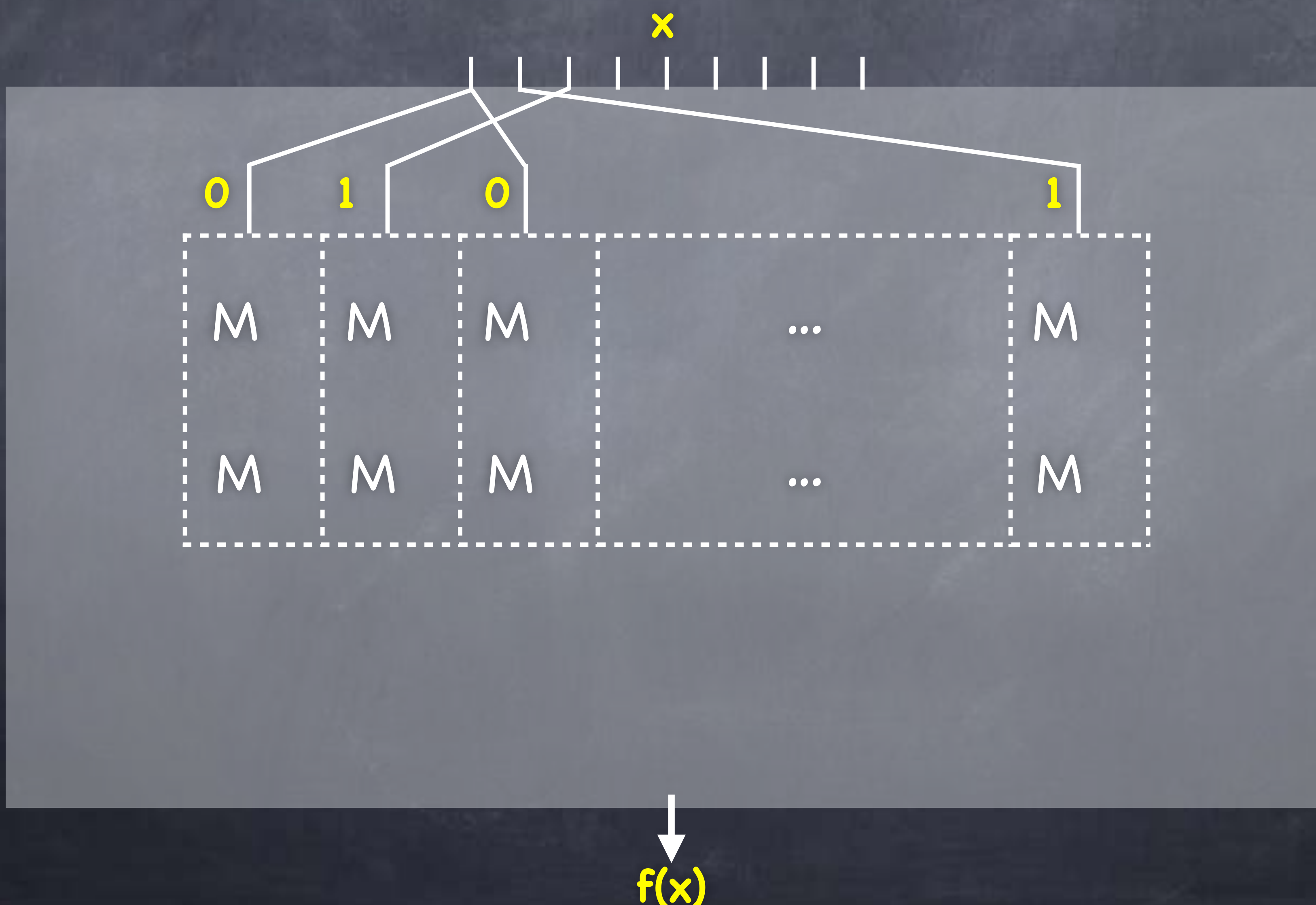
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))



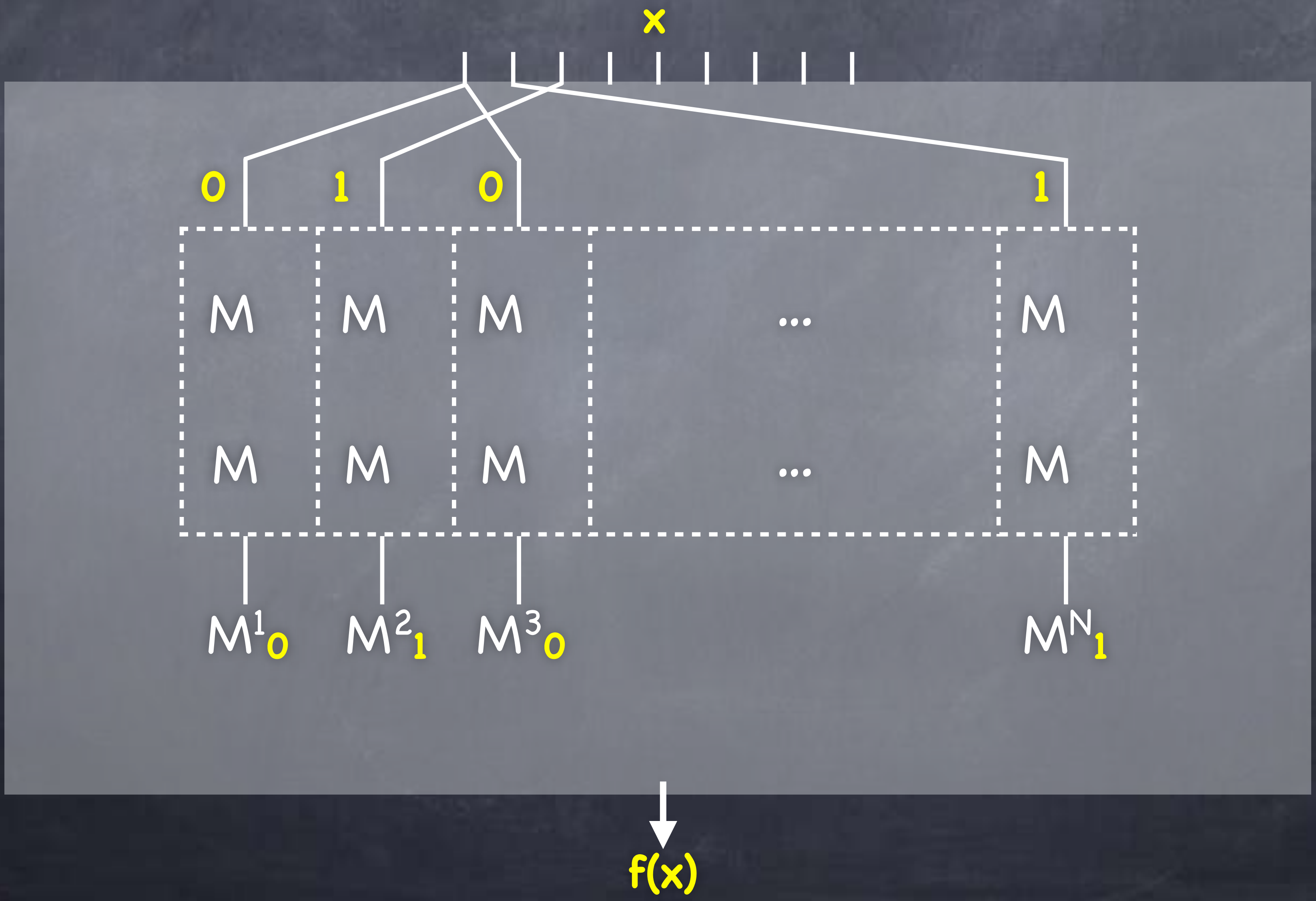
Matrix Programs

- $f : \{0,1\}^n \to \{0,1\}$ using a set of 2N $w \times w$ matrices ($N = \text{poly}(n)$)



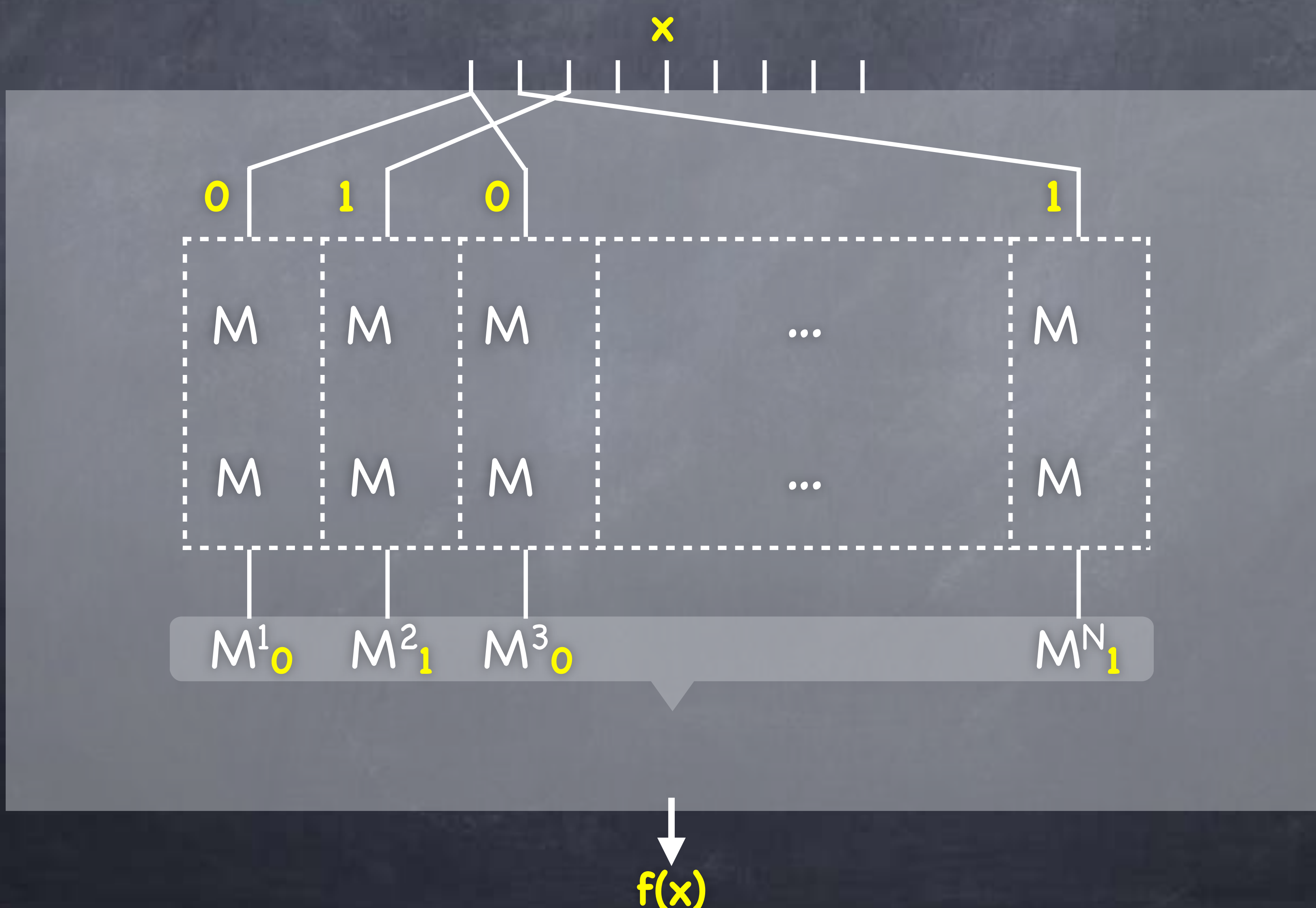
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))



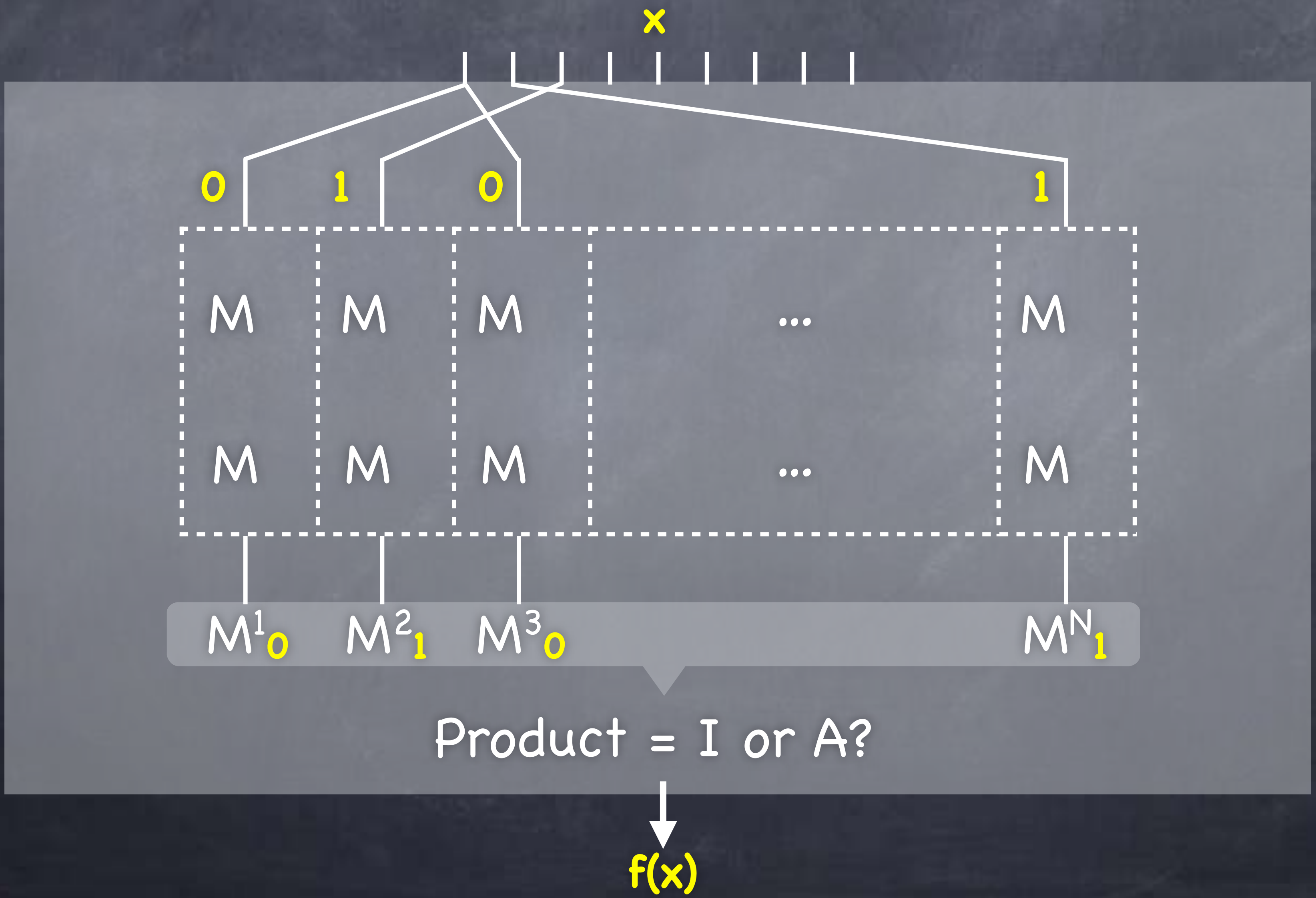
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))



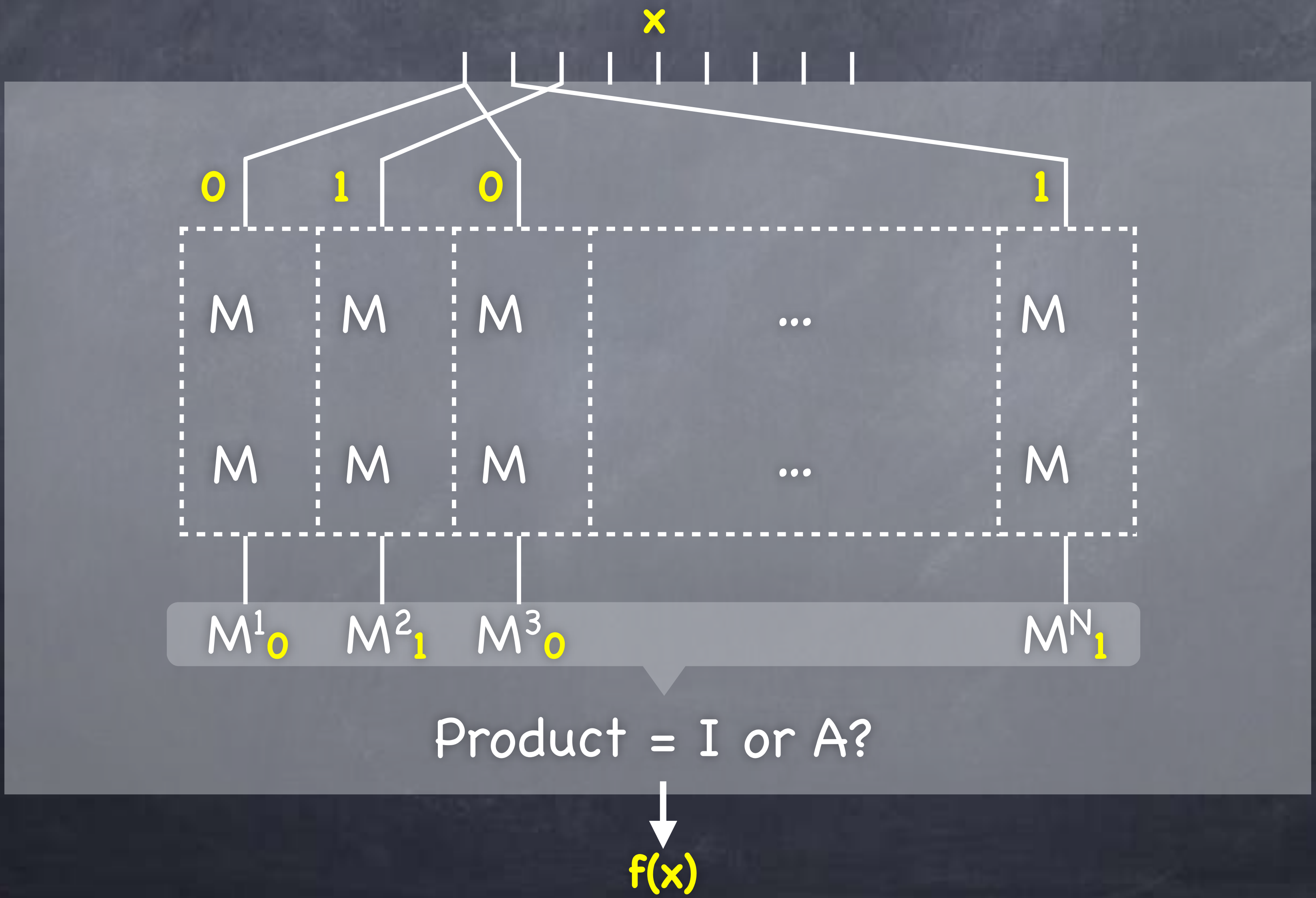
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))



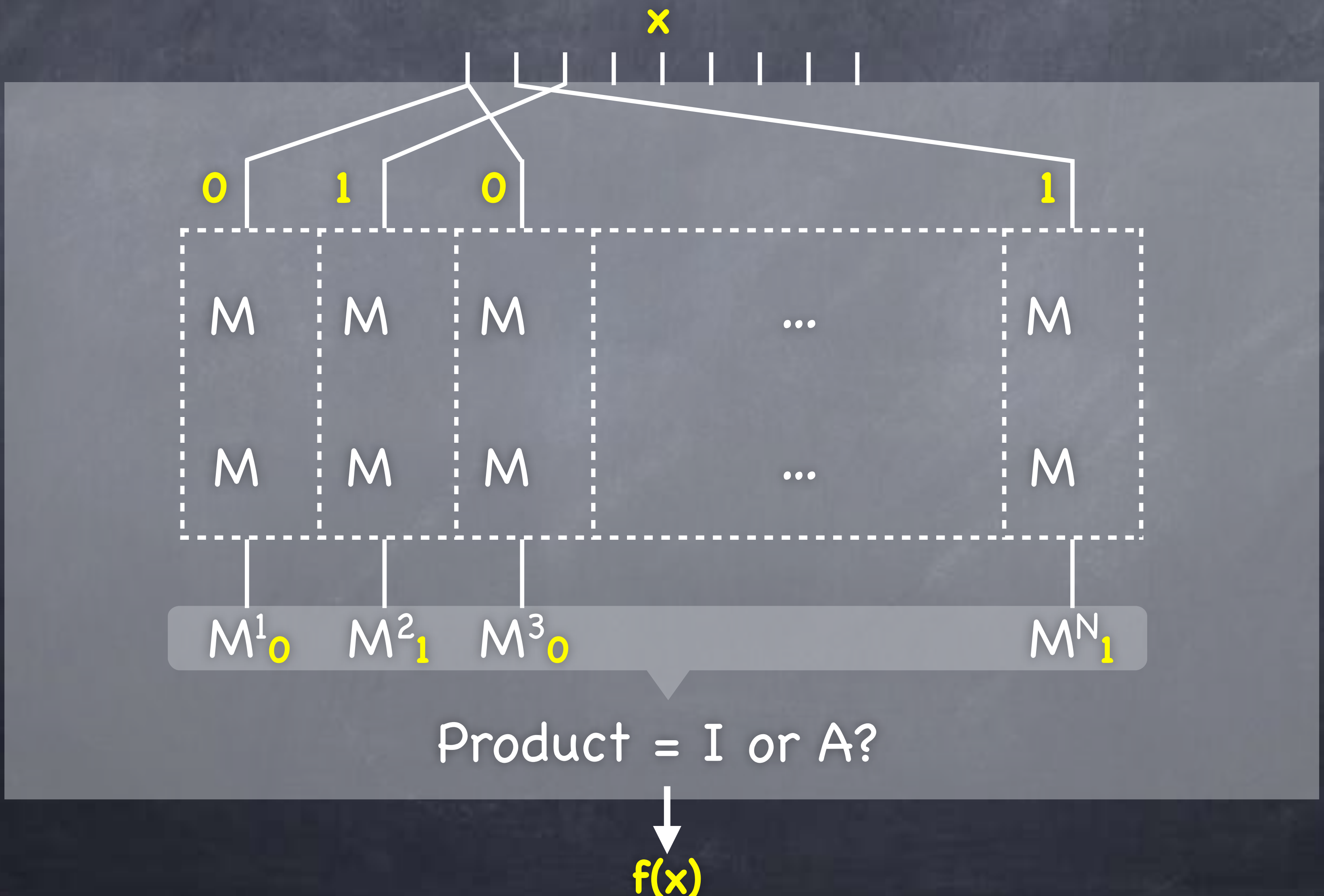
Matrix Programs

- $f : \{0,1\}^n \to \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))
- Family F: all f in F have the same N, w, matrix A and "wiring"



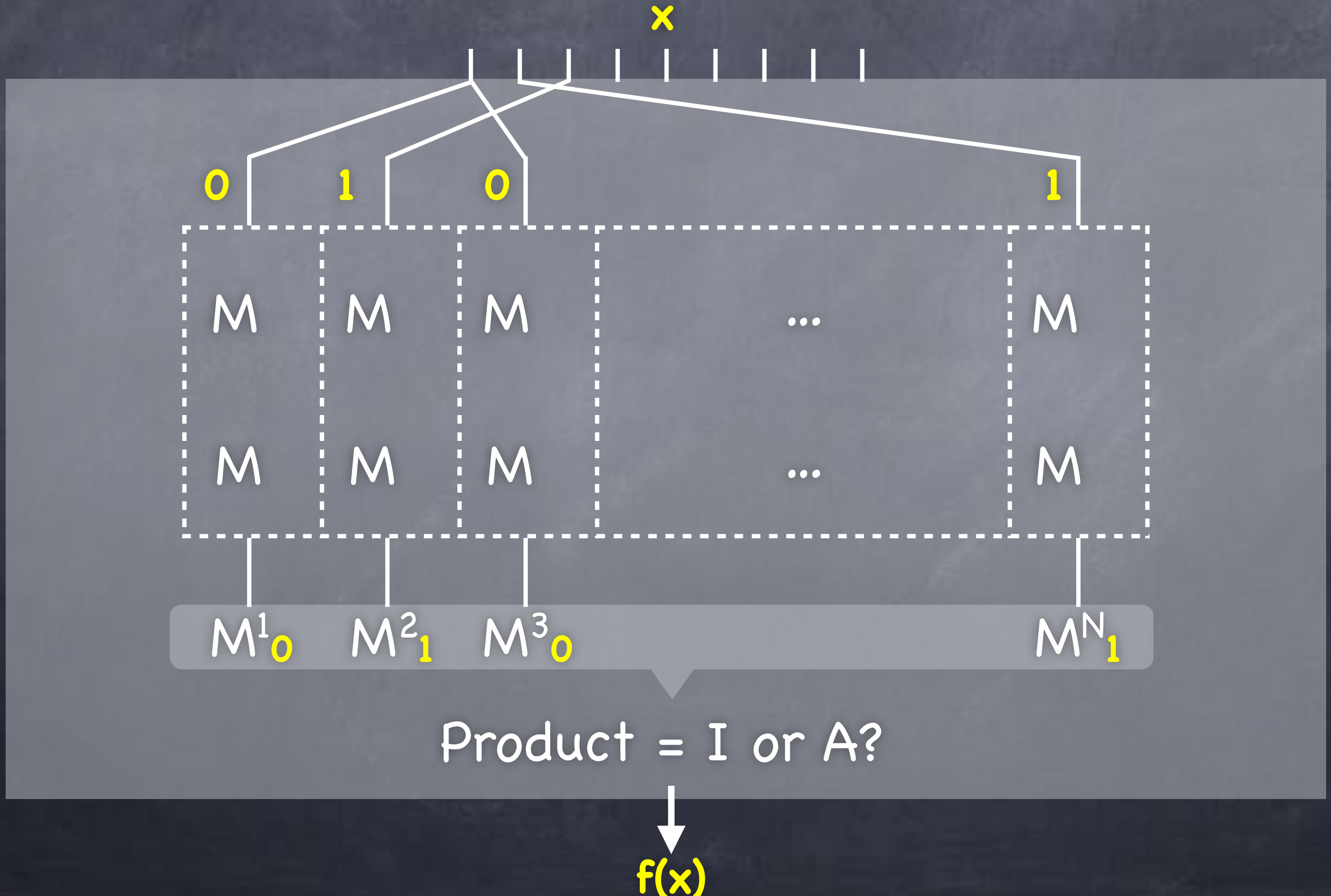
Matrix Programs

- To obfuscate, encode matrices s.t. only valid matrix multiplications and final check can be carried out (for any x)



Matrix Programs

- To obfuscate, encode matrices s.t. only valid matrix multiplications and final check can be carried out (for any x)
- No other information about the 2N matrices should be deducible



Multi-Linear Map

Multi-Linear Map

- Recall groups with bilinear pairing:

Multi-Linear Map

- Recall groups with bilinear pairing:
  - $e: G_1 \times G_2 \rightarrow G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$

Multi-Linear Map

- Recall groups with bilinear pairing:
  - $e: G_1 \times G_2 \rightarrow G_T$ such that $e(g_1{}^a, g_2{}^b) = g_T{}^{ab}$
  - Also group operations in G_i

Multi-Linear Map

- Recall groups with bilinear pairing:
  - e: $G_1 \times G_2 \to G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
  - Also group operations in G_i
  - I.e., one multiplication and several additions (in the exponent)

Multi-Linear Map

- Recall groups with bilinear pairing:
  - e: $G_1 \times G_2 \rightarrow G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
  - Also group operations in G_i
  - I.e., one multiplication and several additions (in the exponent)
  - Assumption: Hard to carry out other operations like $(g_1^a, g_1^b) \mapsto g_T^{ab}$. Heuristic: the Generic Group Model

Multi-Linear Map

- Recall groups with bilinear pairing:
  - e: $G_1 \times G_2 \to G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
  - Also group operations in G_i
  - I.e., one multiplication and several additions (in the exponent)
  - Assumption: Hard to carry out other operations like $(g_1^a, g_1^b) \mapsto g_T^{ab}$. Heuristic: the Generic Group Model
- Extension to more than 2 groups?

Multi-Linear Map

- Recall groups with bilinear pairing:
  - $e: G_1 \times G_2 \to G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
  - Also group operations in G_i
  - I.e., one multiplication and several additions (in the exponent)
  - Assumption: Hard to carry out other operations like $(g_1^a, g_1^b) \mapsto g_T^{ab}$. Heuristic: the Generic Group Model
- Extension to more than 2 groups?
  - Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.

Multi-Linear Map

- Recall groups with bilinear pairing:
  - e: $G_1 \times G_2 \to G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
  - Also group operations in G_i
  - I.e., one multiplication and several additions (in the exponent)
  - Assumption: Hard to carry out other operations like $(g_1^a, g_1^b) \mapsto g_T^{ab}$. Heuristic: the Generic Group Model
- Extension to more than 2 groups?
  - Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
  - $e(g_{S1}^a, g_{S2}^b) = g_{S3}^{ab}$, where $S_1 \cap S_2 = \emptyset$ and $S_3 = S_1 \cup S_2$

Multi-Linear Map

Multi-Linear Map

- Let $T = \{1,\dots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.

Multi-Linear Map

- Let T = {1,…,k}. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think g_S^a)

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think $g_S{}^a$)
  - Need a private key for encoding (think of keeping g_S secret)

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think $g_S{}^a$)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think g_S^a)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$
- Following public operations:

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think $g_S{}^a$)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$
- Following public operations:
  - $[a]_S + [b]_S \to [a+b]_S$ (note that S is the same for all)

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think g_S^a)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$
- Following public operations:
  - $[a]_S + [b]_S \rightarrow [a+b]_S$ (note that S is the same for all)
  - $[a]_{S1} * [b]_{S2} \rightarrow [ab]_{S1 \cup S2}$ where $S_1 \cap S_2 = \varnothing$ and $S_3 = S_1 \cup S_2$

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think g_S^a)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$
- Following public operations:
  - $[a]_S + [b]_S \rightarrow [a+b]_S$ (note that S is the same for all)
  - $[a]_{S1} * [b]_{S2} \rightarrow [ab]_{S1 \cup S2}$ where $S_1 \cap S_2 = \emptyset$ and $S_3 = S_1 \cup S_2$
  - Zero-Test($[a]_T$) checks if a=0 or not (note: only for set T)

Multi-Linear Map

- Let $T = \{1,\ldots,k\}$. For each non-empty subset $S \subseteq T$, a group G_S.
- An element a encoded in G_S: $[a]_S$ (think g_S^a)
  - Need a private key for encoding (think of keeping g_S secret)
  - Allowed to learn the set S from $[a]_S$
- Following public operations:
  - $[a]_S + [b]_S \rightarrow [a+b]_S$ (note that S is the same for all)
  - $[a]_{S1} * [b]_{S2} \rightarrow [ab]_{S1 \cup S2}$ where $S_1 \cap S_2 = \varnothing$ and $S_3 = S_1 \cup S_2$
  - Zero-Test($[a]_T$) checks if a=0 or not (note: only for set T)
- Generic Group Model heuristic: No other operation possible!

Obfuscation from Multi-Linear Map

Obfuscation from Multi-Linear Map

- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements

Obfuscation from Multi-Linear Map

- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements
    - Final outcome checked as $[a]_T = [v]_T$, where $[a]_T$ is computed and $[v]_T$ is included as part of the obfuscation

Obfuscation from Multi-Linear Map

- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements
  - Final outcome checked as $[a]_T = [v]_T$, where $[a]_T$ is computed and $[v]_T$ is included as part of the obfuscation
- Each matrix encoded using an associated set S

Obfuscation from Multi-Linear Map

- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements
  - Final outcome checked as $[a]_T = [v]_T$, where $[a]_T$ is computed and $[v]_T$ is included as part of the obfuscation
- Each matrix encoded using an associated set S
  - Sets chosen so as to prevent invalid combinations

Obfuscation from Multi-Linear Map

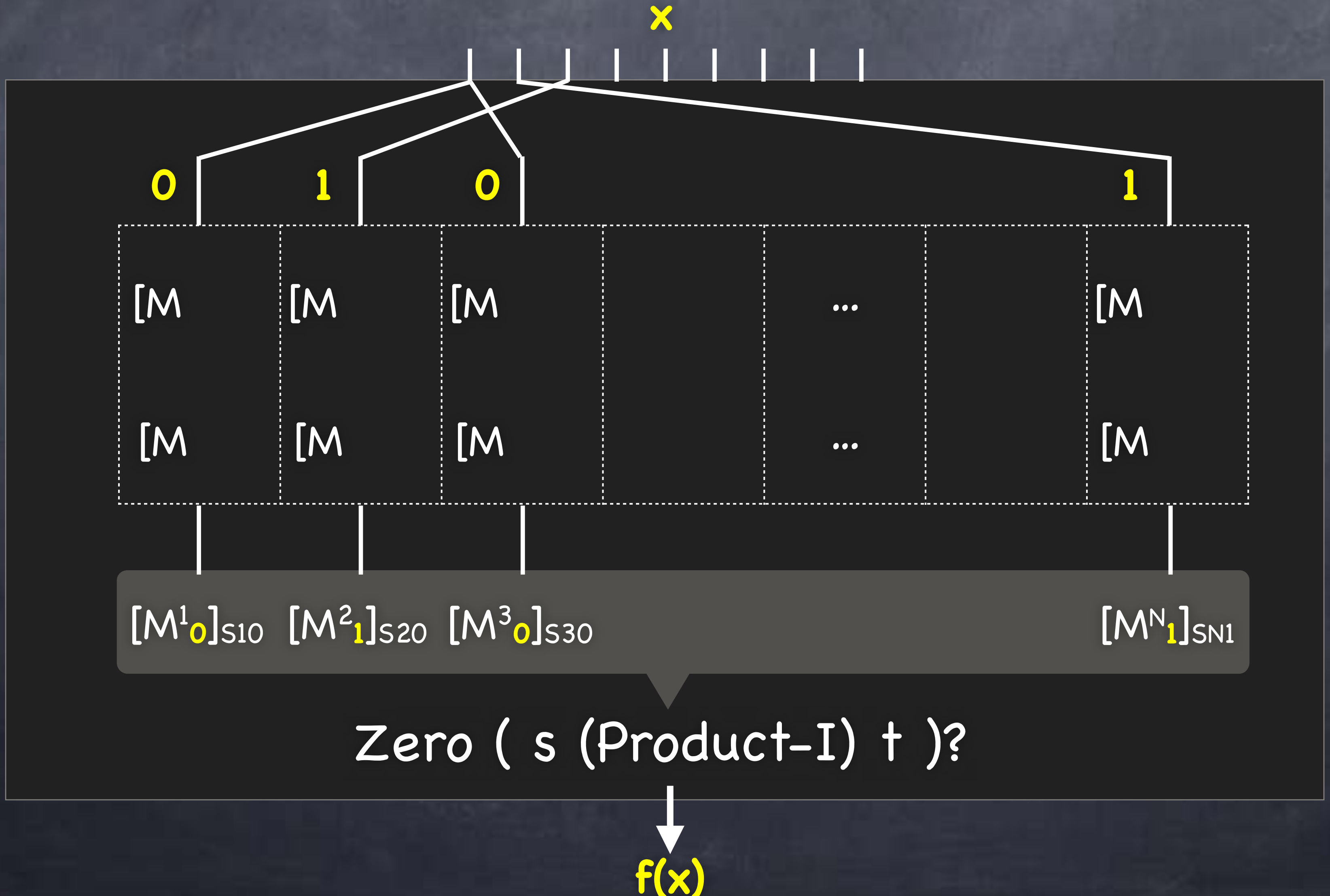
- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements
  - Final outcome checked as $[a]_T = [v]_T$, where $[a]_T$ is computed and $[v]_T$ is included as part of the obfuscation
- Each matrix encoded using an associated set S
  - Sets chosen so as to prevent invalid combinations
  - Matrices randomized (while preserving product) to ensure that the matrices cannot be reordered/tampered with

Obfuscation from Multi-Linear Map

- Matrix elements are encoded using the multi-linear map, so that matrix product can be carried out on encoded elements
  - Final outcome checked as $[a]_T = [v]_T$, where $[a]_T$ is computed and $[v]_T$ is included as part of the obfuscation
- Each matrix encoded using an associated set S
  - Sets chosen so as to prevent invalid combinations
  - Matrices randomized (while preserving product) to ensure that the matrices cannot be reordered/tampered with
    - Any tampering will result (w.h.p.) in $[a]_T$ being random (and independent each time)

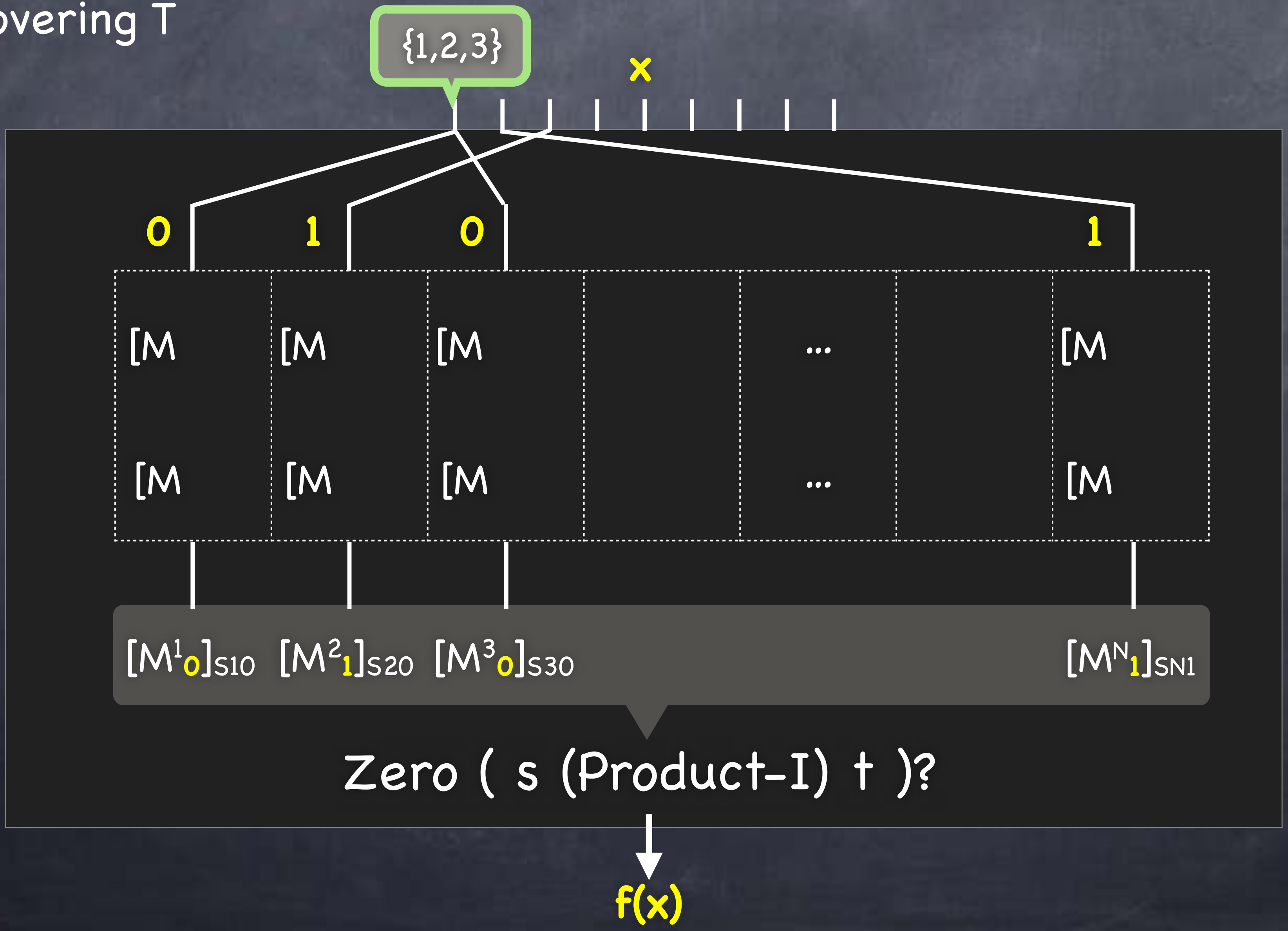
Obfuscating Matrix Programs

- Preventing invalid combinations: entries in $M^i_{0/1}$ encoded for set $S^i_{0/1}$ so that invalid combinations result in intersecting sets, or sets not covering T



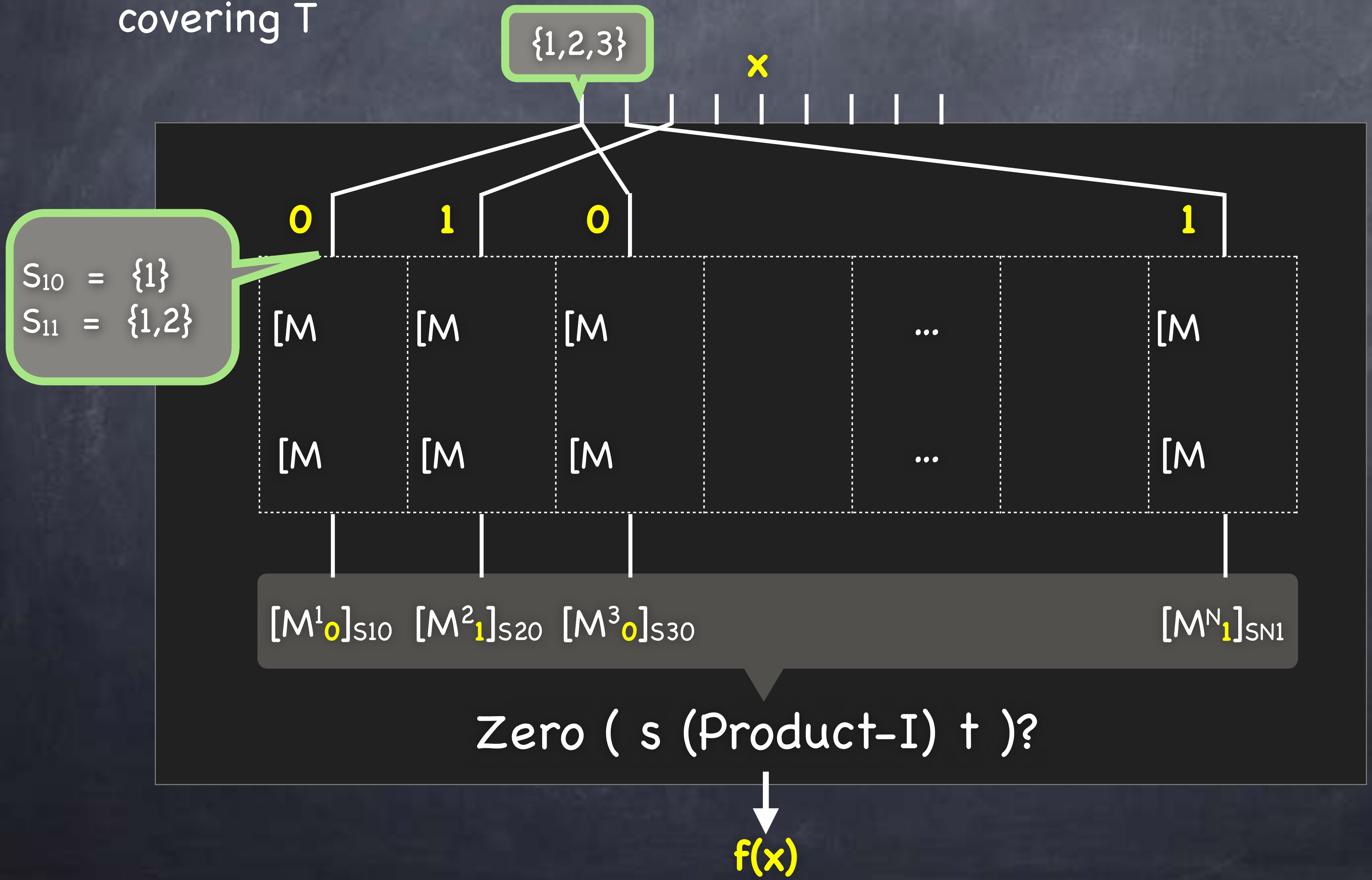
Obfuscating Matrix Programs

- Preventing invalid combinations: entries in $M^i_{0/1}$ encoded for set $S^i_{0/1}$ so that invalid combinations result in intersecting sets, or sets not covering T



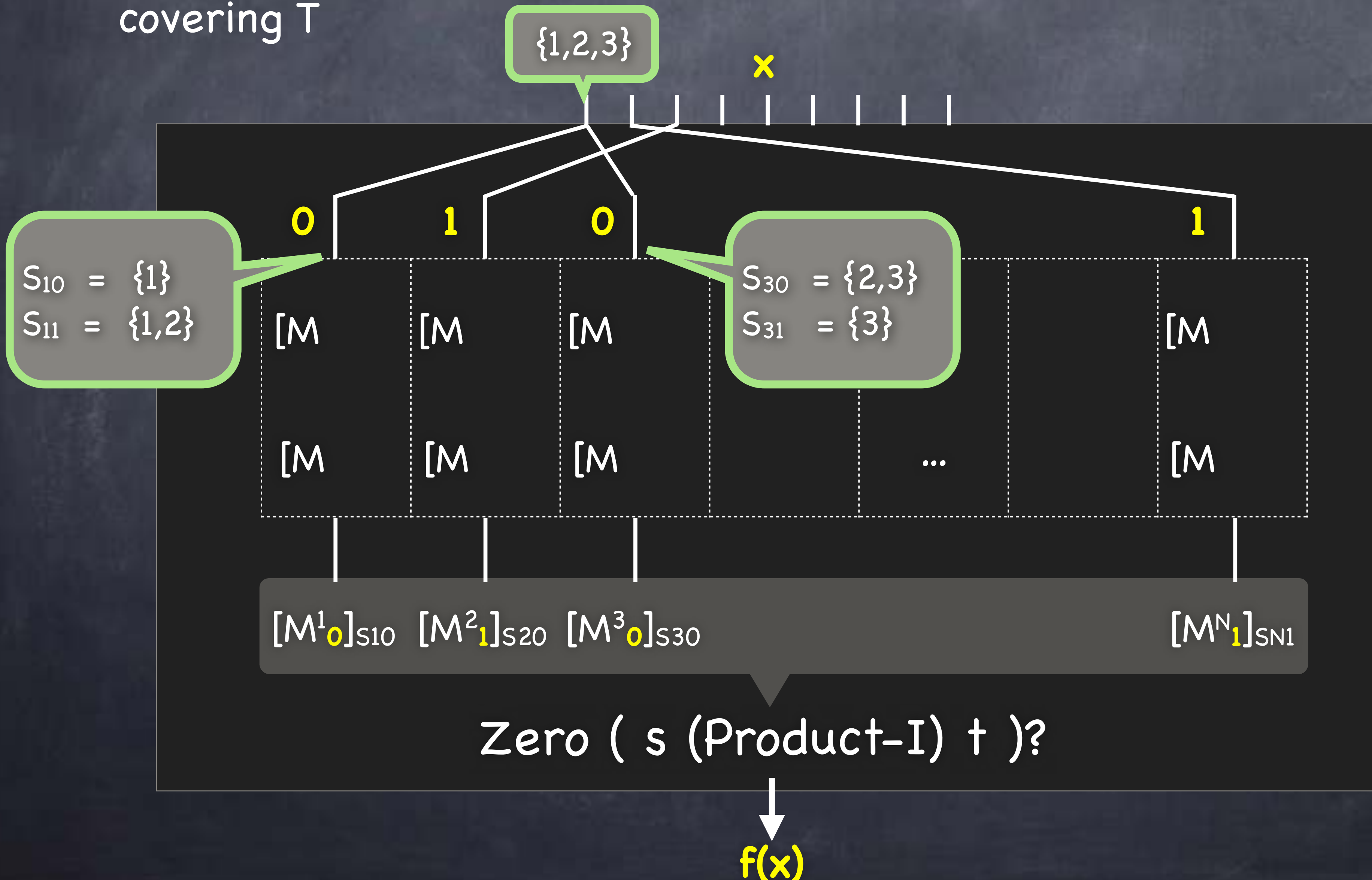
Obfuscating Matrix Programs

- Preventing invalid combinations: entries in $M^i_{0/1}$ encoded for set $S^i_{0/1}$ so that invalid combinations result in intersecting sets, or sets not covering T



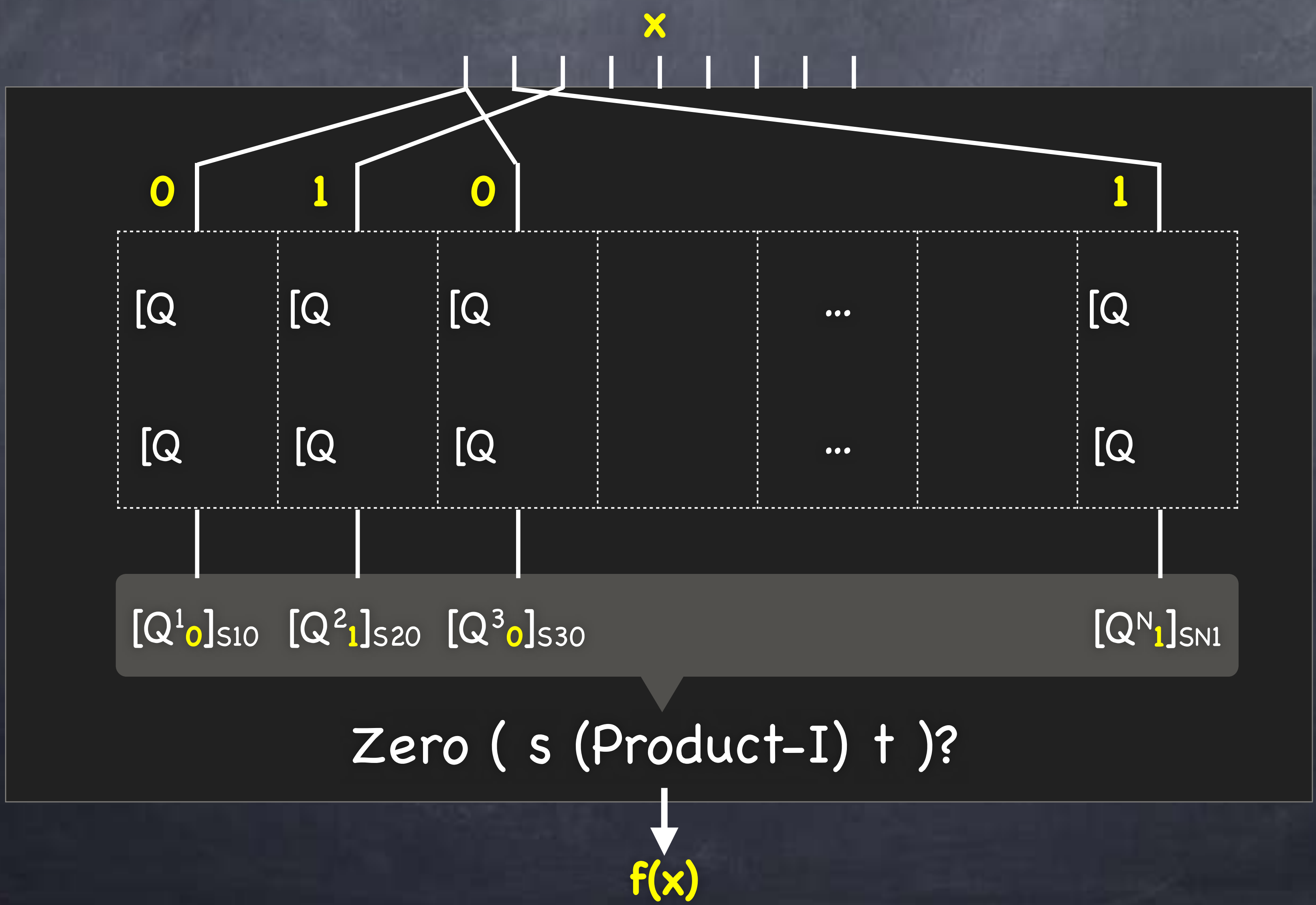
Obfuscating Matrix Programs

- Preventing invalid combinations: entries in $M^i_{0/1}$ encoded for set $S^i_{0/1}$ so that invalid combinations result in intersecting sets, or sets not covering T



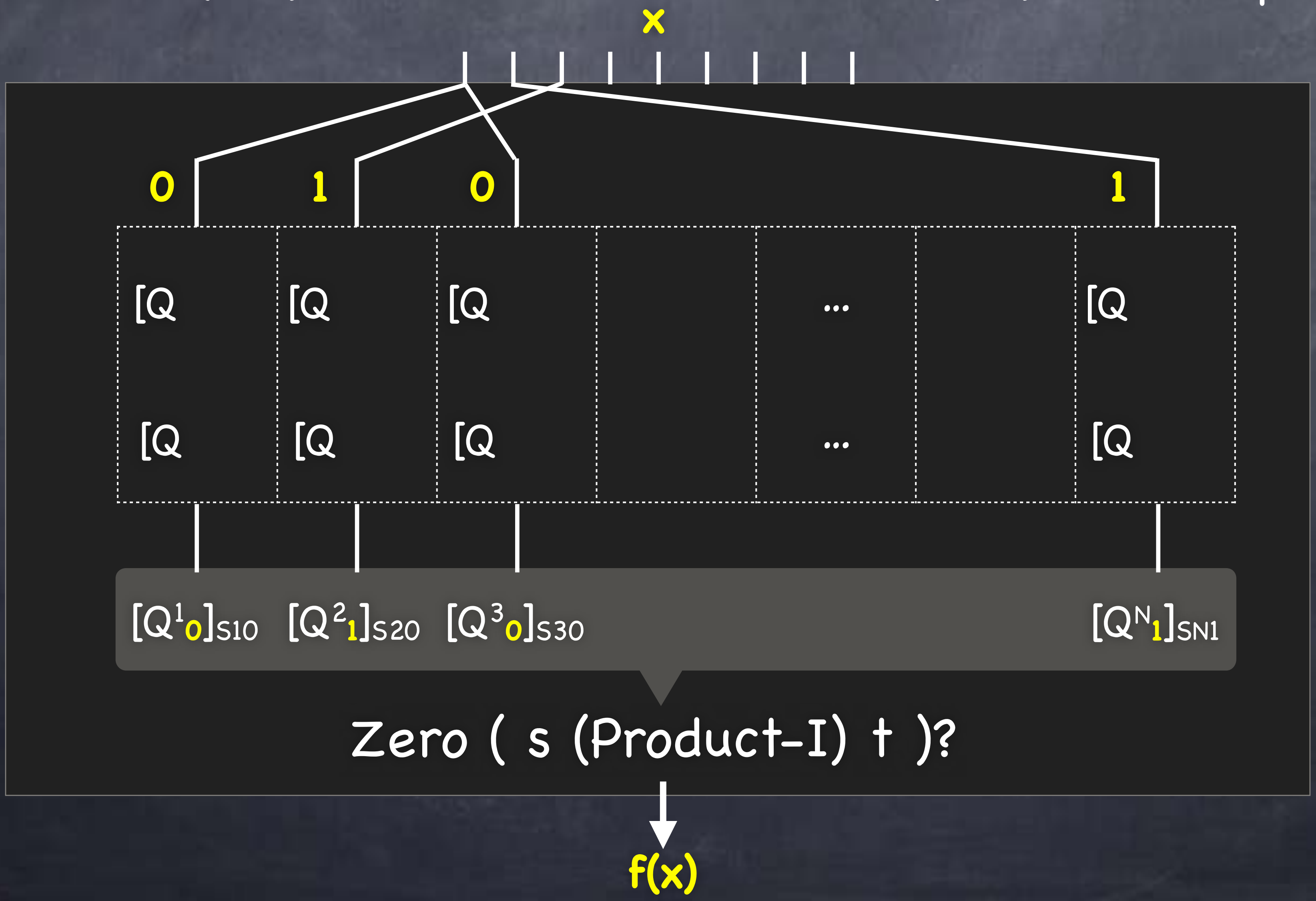
Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices



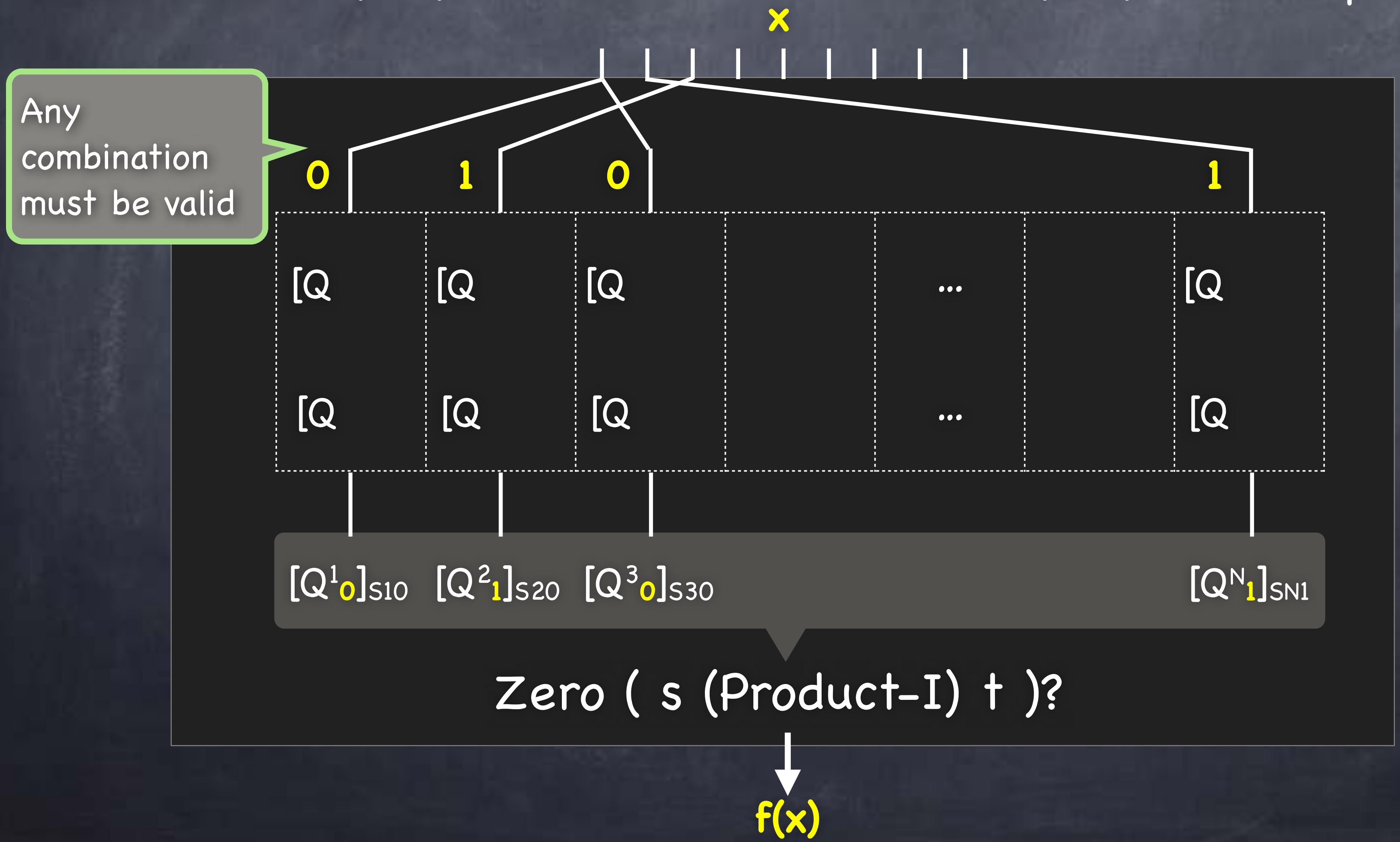
Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices
  - Let $Q^i_b = R_{i-1} M^i_b R_i^{-1}$ (R_i random, $R_0=R_N=I$): $\prod_i Q^i_{bi} = \prod_i M^i_{bi}$ while $\{Q^i_{bi}\}$ has no information about $\{M^i_{bi}\}$ than its product



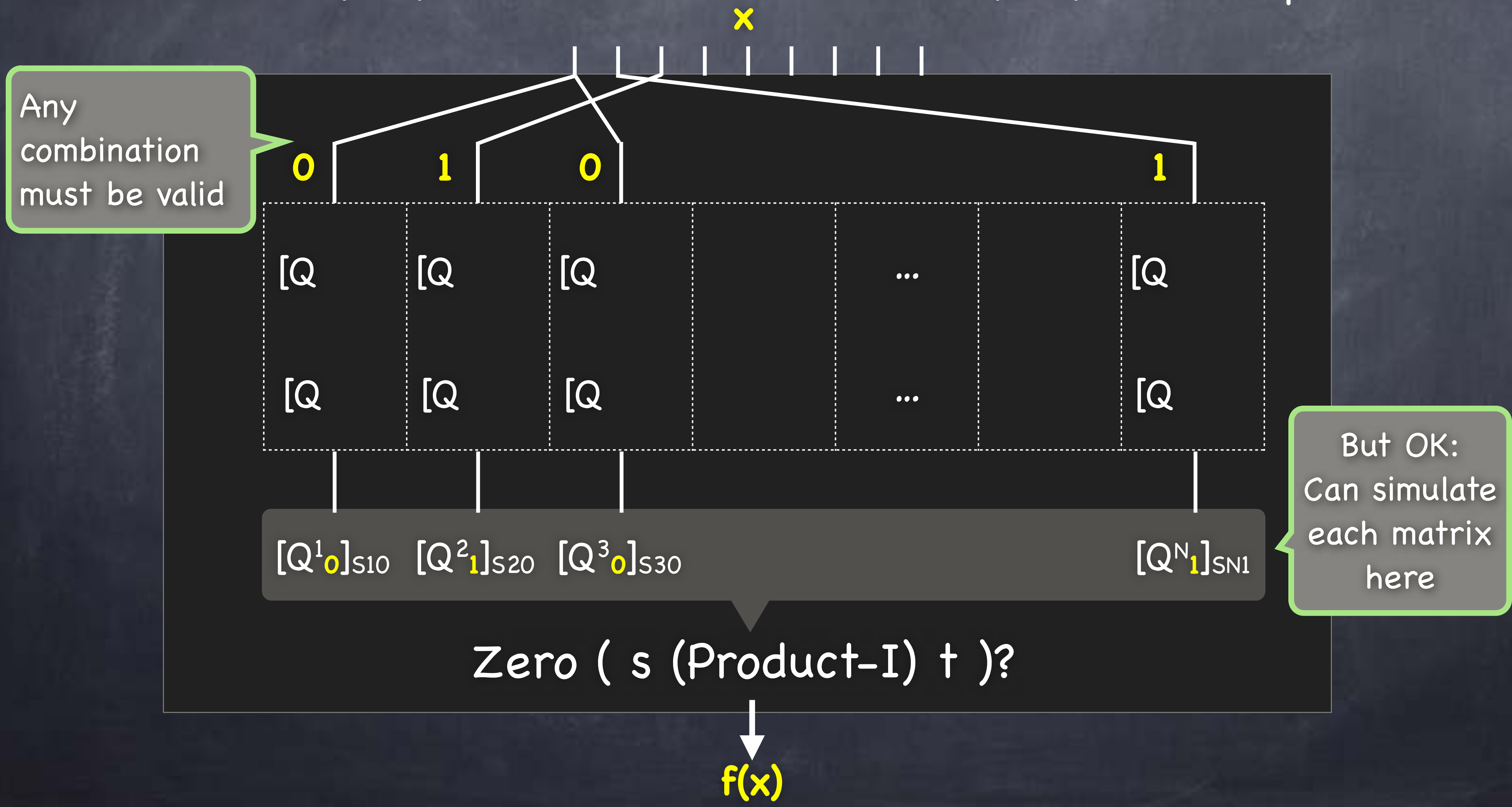
Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices
  - Let $Q^i_b = R_{i-1} M^i_b R_i^{-1}$ (R_i random, $R_0=R_N=I$): $\prod_i Q^i_{bi} = \prod_i M^i_{bi}$ while $\{Q^i_{bi}\}$ has no information about $\{M^i_{bi}\}$ than its product



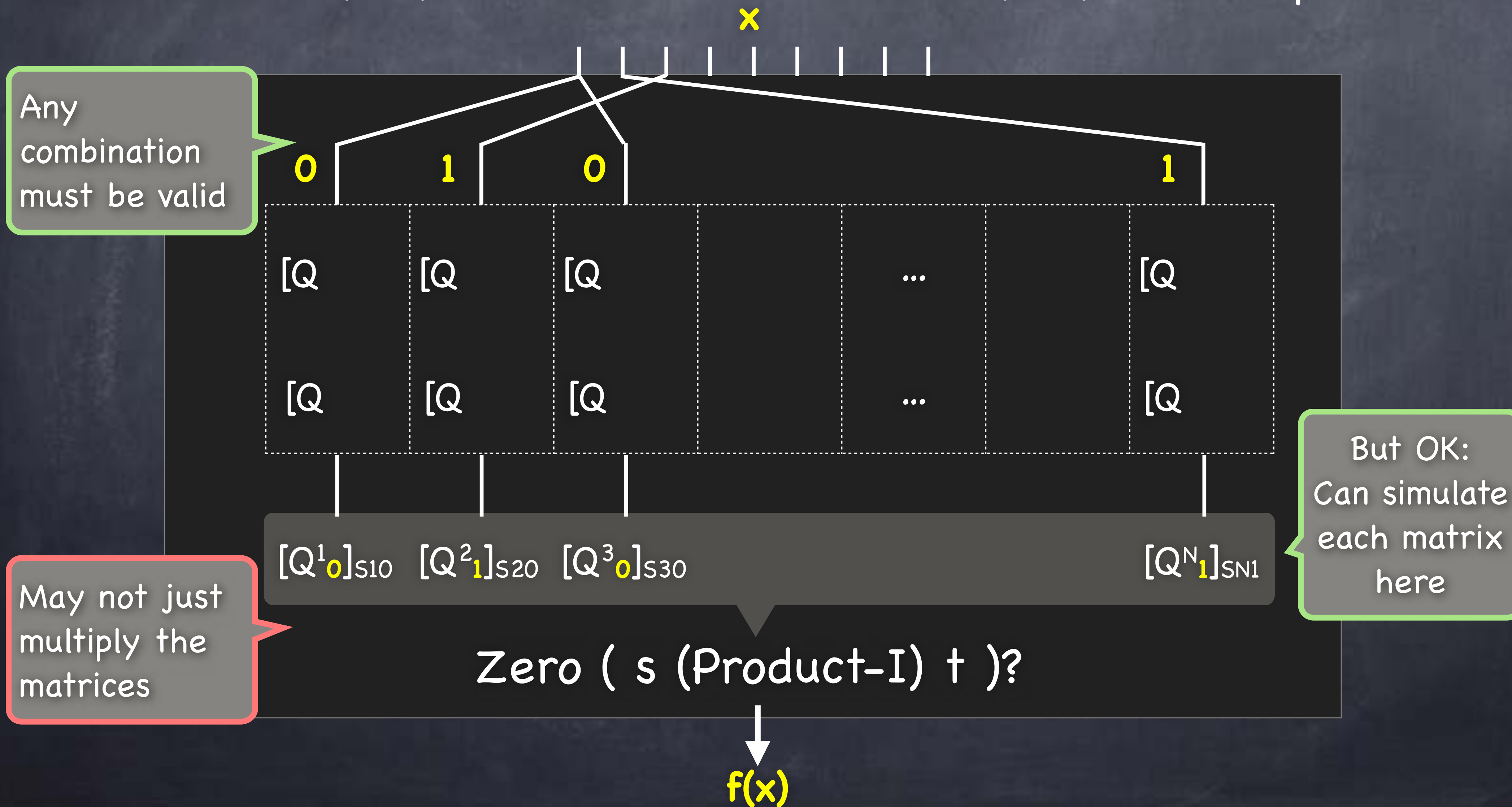
Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices
  - Let $Q^i_b = R_{i-1} M^i_b R_i^{-1}$ (R_i random, $R_0=R_N=I$): $\prod_i Q^i_{b_i} = \prod_i M^i_{b_i}$ while $\{Q^i_{b_i}\}$ has no information about $\{M^i_{b_i}\}$ than its product



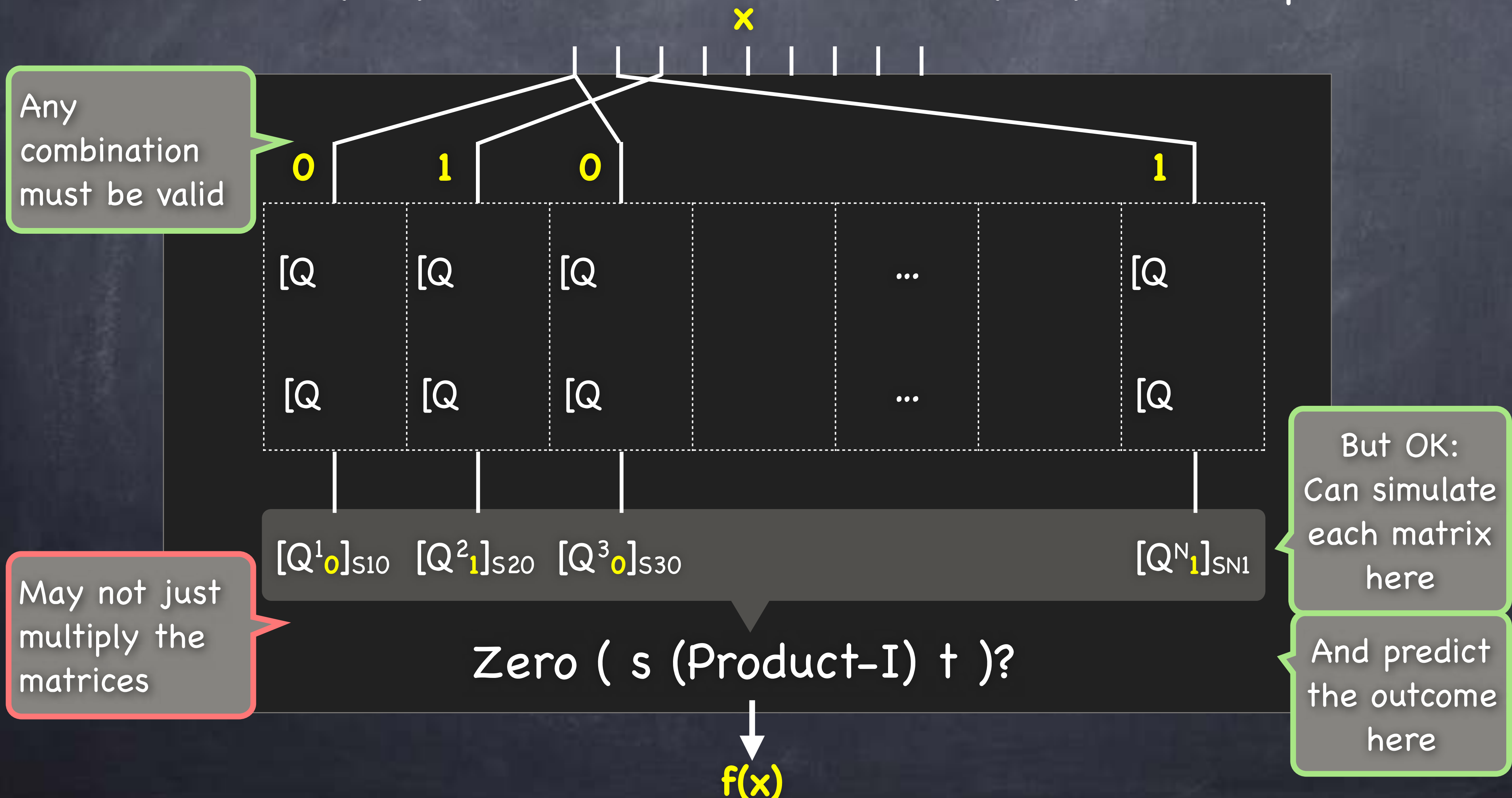
Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices
  - Let $Q^i_b = R_{i-1} M^i_b R_i^{-1}$ (R_i random, $R_0=R_N=I$): $\Pi_i Q^i_{bi} = \Pi_i M^i_{bi}$ while $\{Q^i_{bi}\}$ has no information about $\{M^i_{bi}\}$ than its product



Obfuscating Matrix Programs

- Ensure no information by reordering/tampering with the matrices
  - Let $Q^i_b = R_{i-1} M^i_b R_i^{-1}$ (R_i random, $R_0=R_N=I$): $\prod_i Q^i_{b_i} = \prod_i M^i_{b_i}$ while $\{Q^i_{b_i}\}$ has no information about $\{M^i_{b_i}\}$ than its product



Obfuscating Matrix Programs

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 matrices)

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 matrices)
    - Can "bootstrap" from this to all polynomial-sized circuits/ polynomial-time computable functions, assuming "Fully Homomorphic Encryption" (with decryption in NC^1)

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 matrices)
    - Can "bootstrap" from this to all polynomial-sized circuits/polynomial-time computable functions, assuming "Fully Homomorphic Encryption" (with decryption in NC^1)
- Do multi-linear maps exist?

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 matrices)
    - Can "bootstrap" from this to all polynomial-sized circuits/ polynomial-time computable functions, assuming "Fully Homomorphic Encryption" (with decryption in NC^1)
- Do multi-linear maps exist?
    - Generic multi-linear map model is an unrealizable model (and VBB obfuscation for NC^1 is impossible)

Obfuscating Matrix Programs

- Using generic multi-linear map, can obfuscate polynomial-sized matrix programs: yields Virtual Black-Box obfuscation
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 matrices)
  - Can "bootstrap" from this to all polynomial-sized circuits/polynomial-time computable functions, assuming "Fully Homomorphic Encryption" (with decryption in NC^1)
- Do multi-linear maps exist?
  - Generic multi-linear map model is an unrealizable model (and VBB obfuscation for NC^1 is impossible)
  - Weaker multi-linear maps?

Obfuscating Matrix Programs

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding
  - Certain problems (a la DDH and DBDH) conjectured to be hard

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding
  - Certain problems (a la DDH and DBDH) conjectured to be hard
- Instantiating obfuscation constructions using these candidates yield weaker forms of obfuscation (in standard model)

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding
  - Certain problems (a la DDH and DBDH) conjectured to be hard
- Instantiating obfuscation constructions using these candidates yield weaker forms of obfuscation (in standard model)
- Indistinguishability Obfuscation (iO), Differing-Inputs Obfuscation (DIO), etc.

Obfuscating Matrix Programs

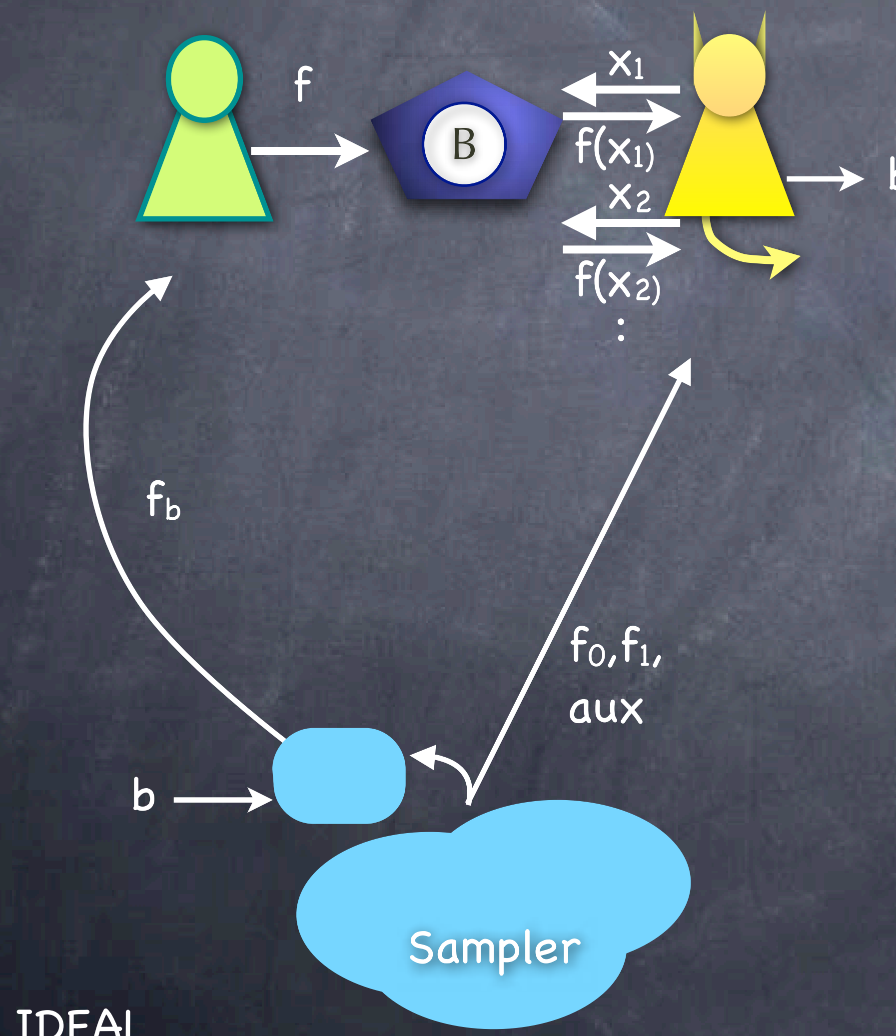
- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding
  - Certain problems (a la DDH and DBDH) conjectured to be hard
- Instantiating obfuscation constructions using these candidates yield weaker forms of obfuscation (in standard model)
- Indistinguishability Obfuscation (iO), Differing-Inputs Obfuscation (DIO), etc.
  - Weaker, but still useful in many applications

Obfuscating Matrix Programs

- Recently, candidate multi-linear maps
  - Based on lattices [GGH'13] and integers [CLT'13]
  - Have noisy, randomized encoding
  - Certain problems (a la DDH and DBDH) conjectured to be hard
- Instantiating obfuscation constructions using these candidates yield weaker forms of obfuscation (in standard model)
- Indistinguishability Obfuscation (iO), Differing-Inputs Obfuscation (DIO), etc.
  - Weaker, but still useful in many applications
  - Security notion: "Indistinguishability-Preserving"

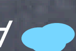
IND-PRE Obfuscation

No simulation of the obfuscated program!
If sampler s.t. b is not hidden in REAL, it must be because b is not hidden in IDEAL
i.e., Hiding in IDEAL ⇒ Hiding in REAL



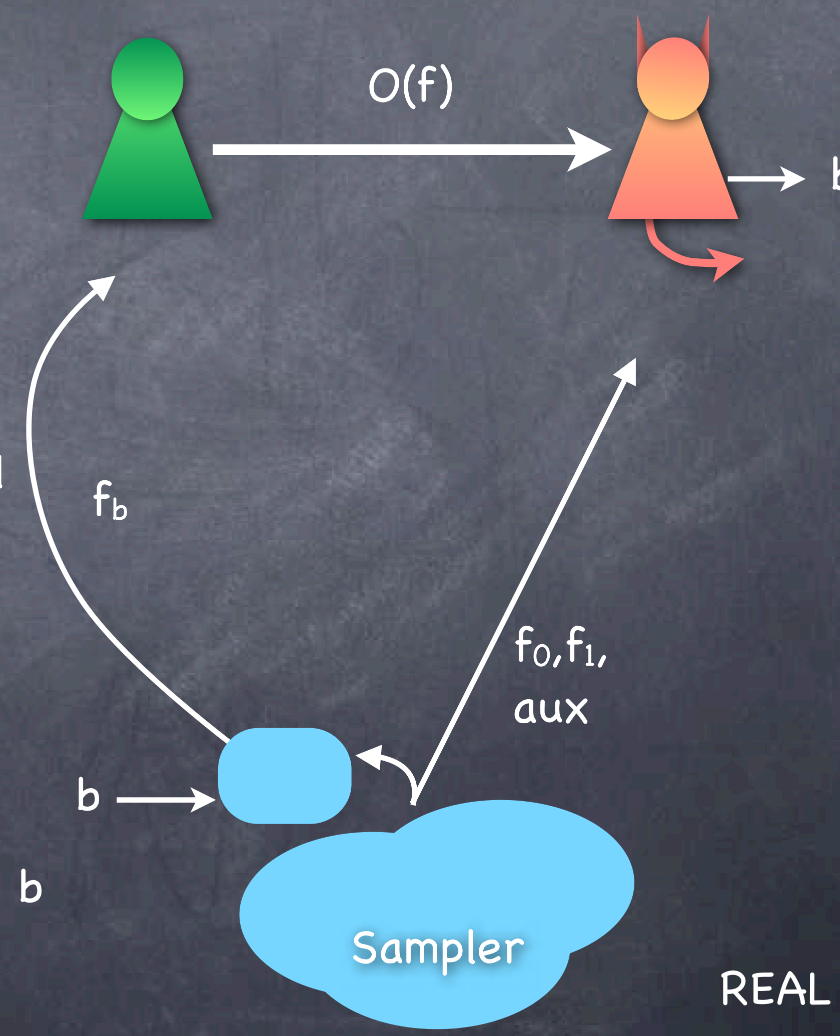
Secure (and correct) if:

∀ 

∀ 

∃  s.t.

if  learns b
so does 



Today

- Obfuscation
- Strong definitions are provably impossible to achieve
- Recent breakthroughs (for weaker definitions)
- Using Multi-linear Maps