



UC What You See: Brave New World of Zero-Overhead Pseudo-Code UC



Markulf Kohlweiss



Jan Bobolz



TCC 2024

ia.cr/2024/818

EUROCRYPT 2026

ia.cr/2026/911

Pooya Farshim



ASIACRYPT 2026

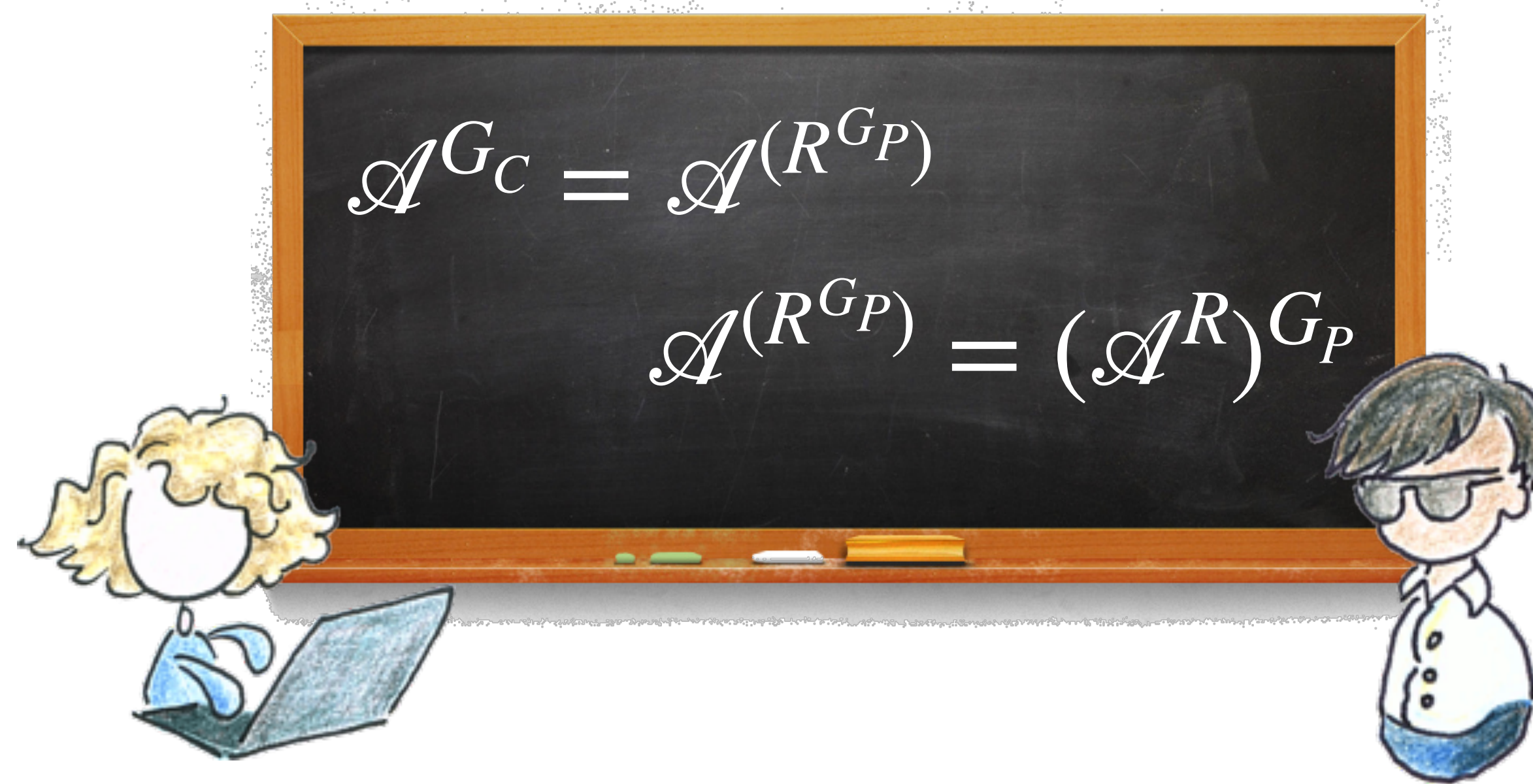
ia.cr/2026/292

Provable Security & Universal Composition

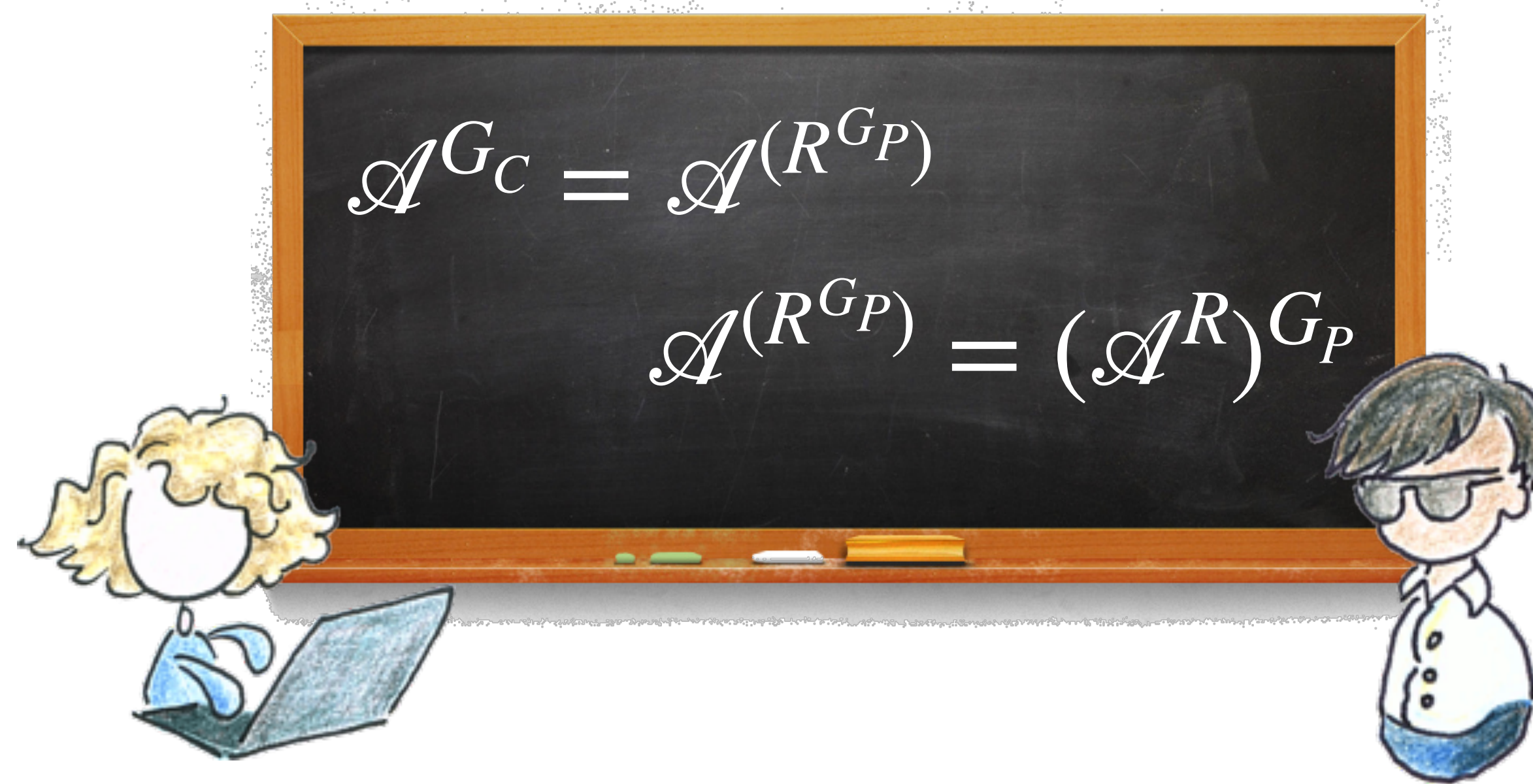
The Good, the Bad, the Ugly



Blackboard view of Provable Security



Blackboard view of Provable Security



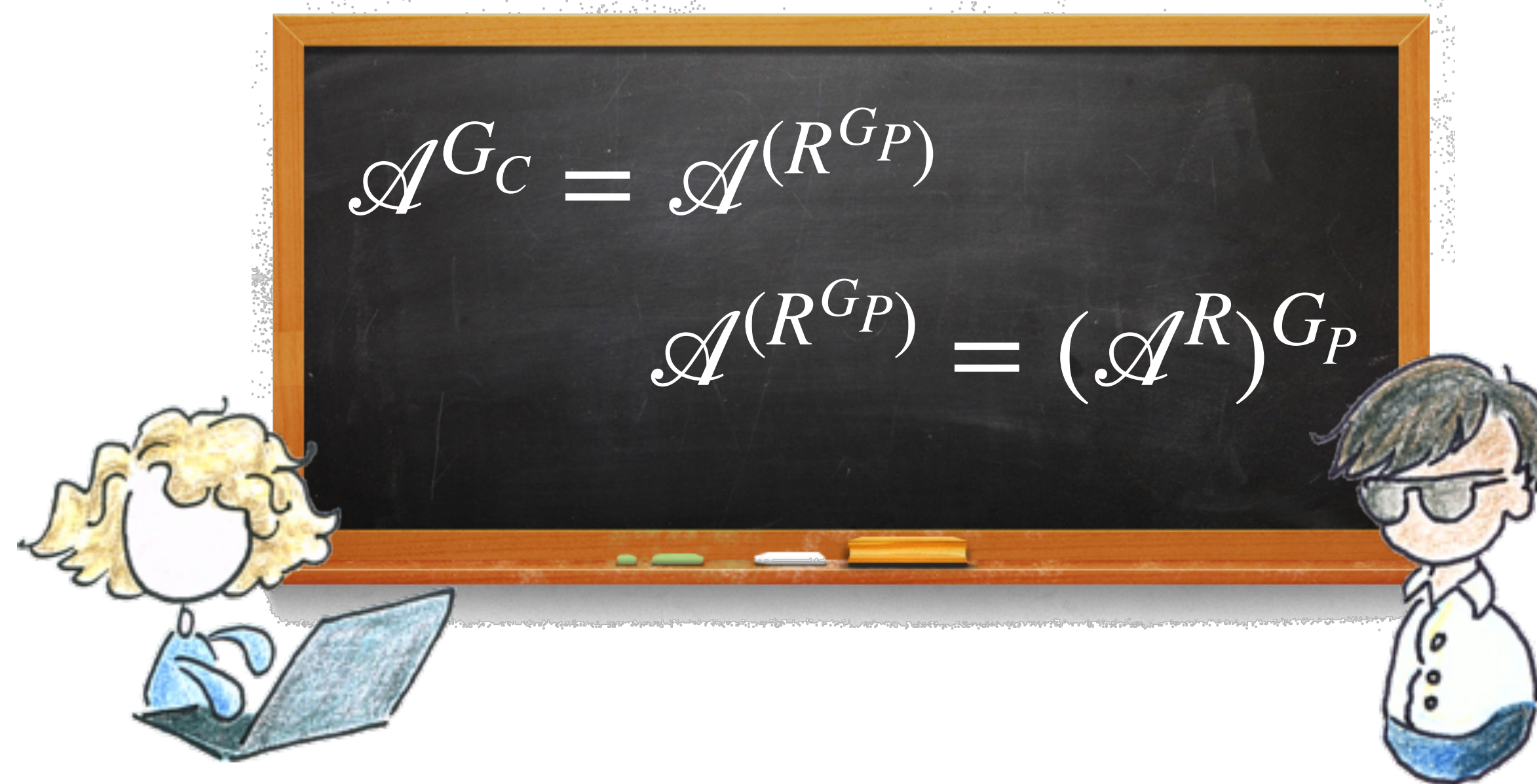
Security reductions are common in cryptography

- Showing soundness of reductions is complicated

State Separating Proofs (SSP)

- is one way of writing reduction proofs in pseudocode

Blackboard view of Provable Security

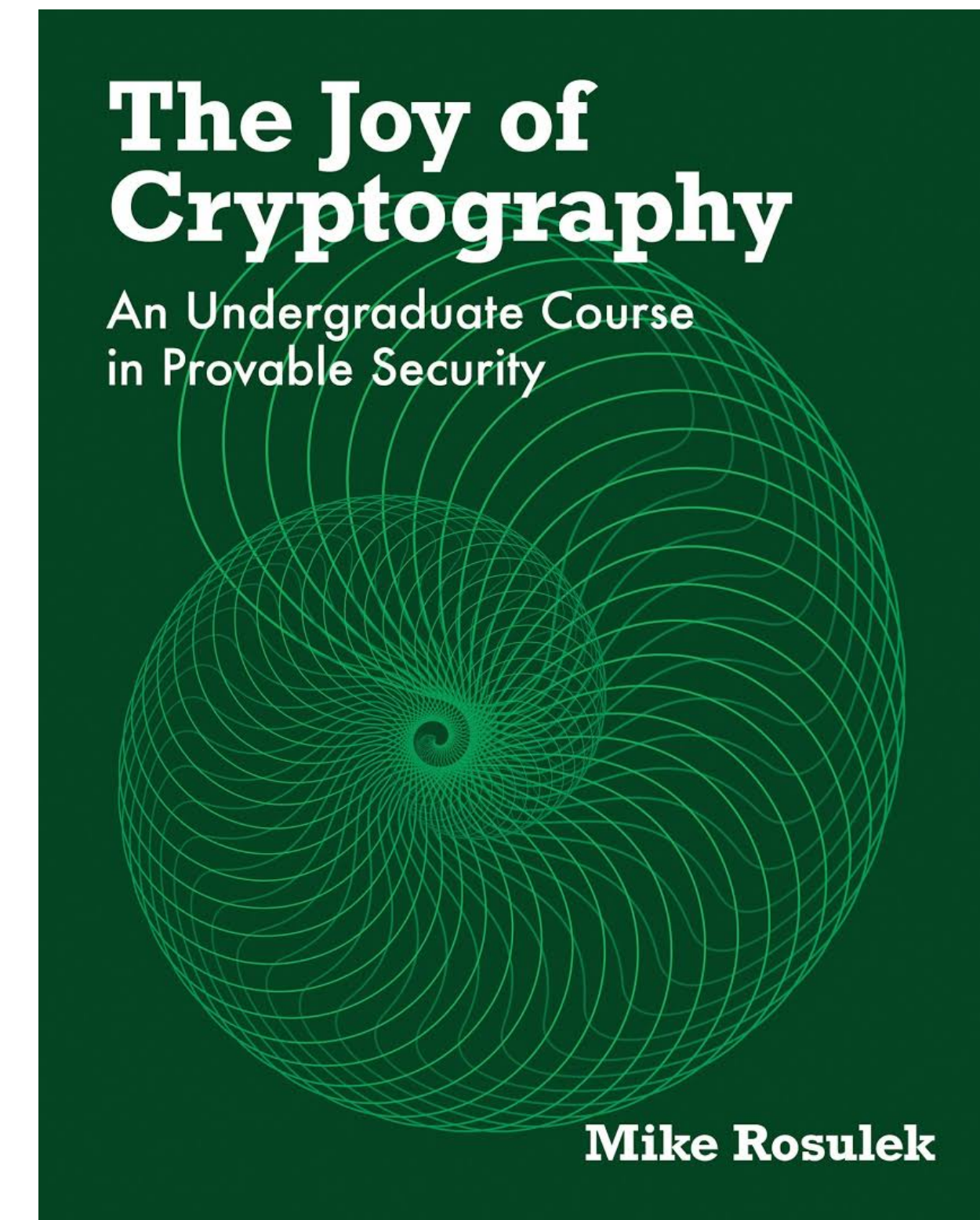


Security reductions are common in cryptography

- Showing soundness of reductions is complicated

State Separating Proofs (SSP)

- is one way of writing reduction proofs in pseudocode

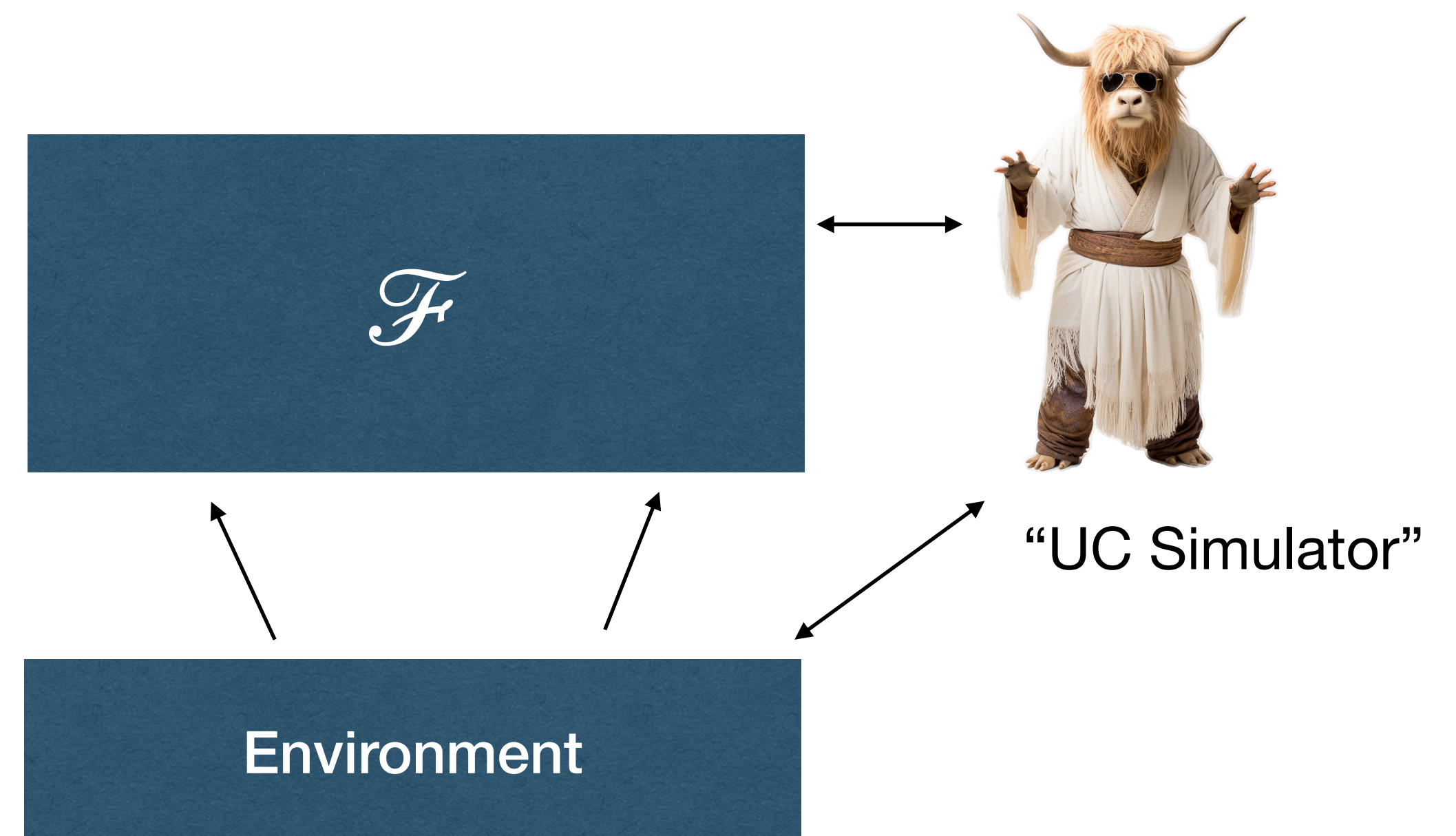


UC modeling

Real world



Ideal world



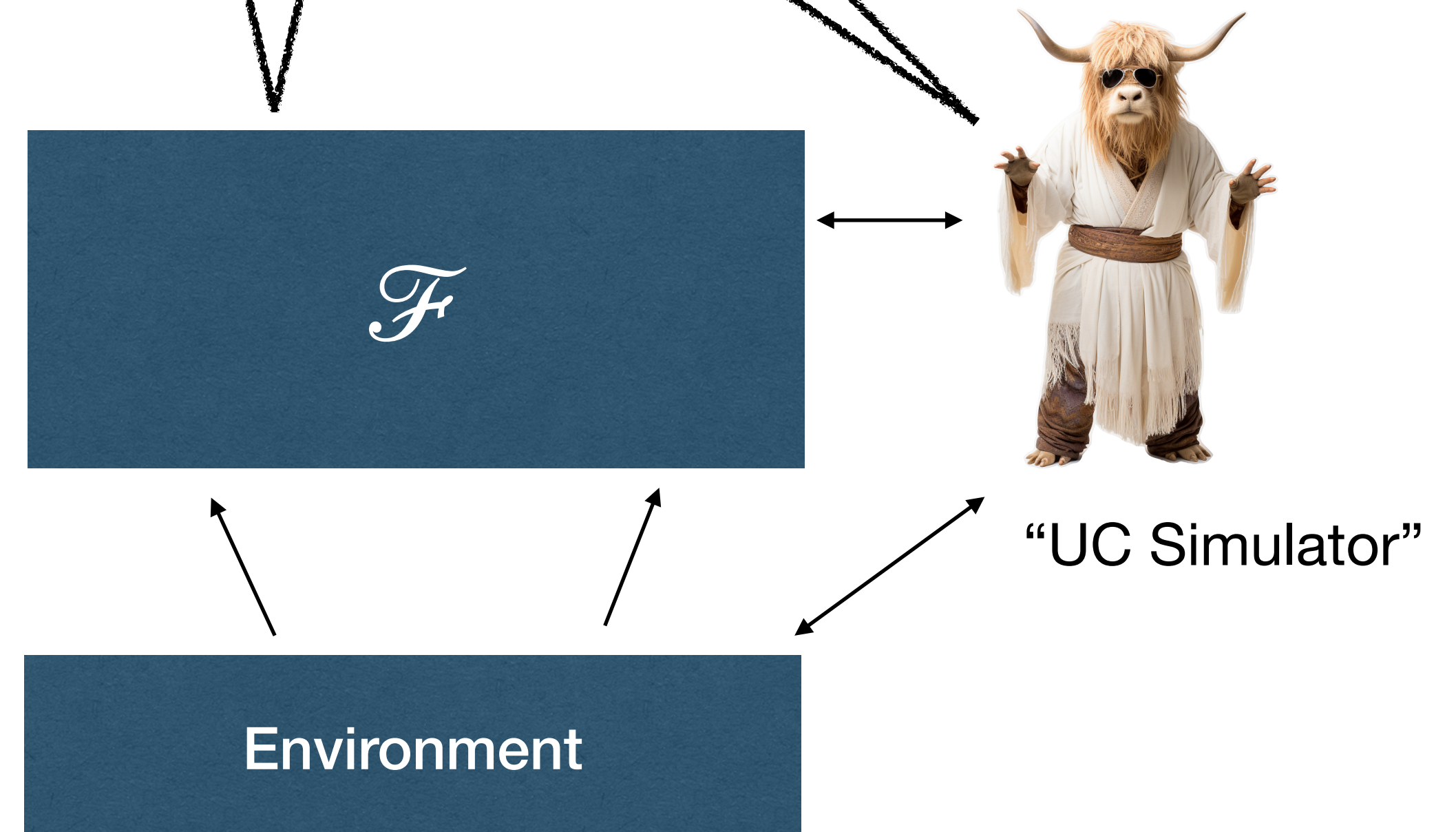
UC modeling

Real world



Ideal world

1. Real protocol *simulated* (using leakage).
2. Adversarial inputs to $\mathcal{F}$ extracted from protocol



The Joy of Universal Composition

An Undergraduate Course
in Writing and Rhetoric

Mike Rosulek

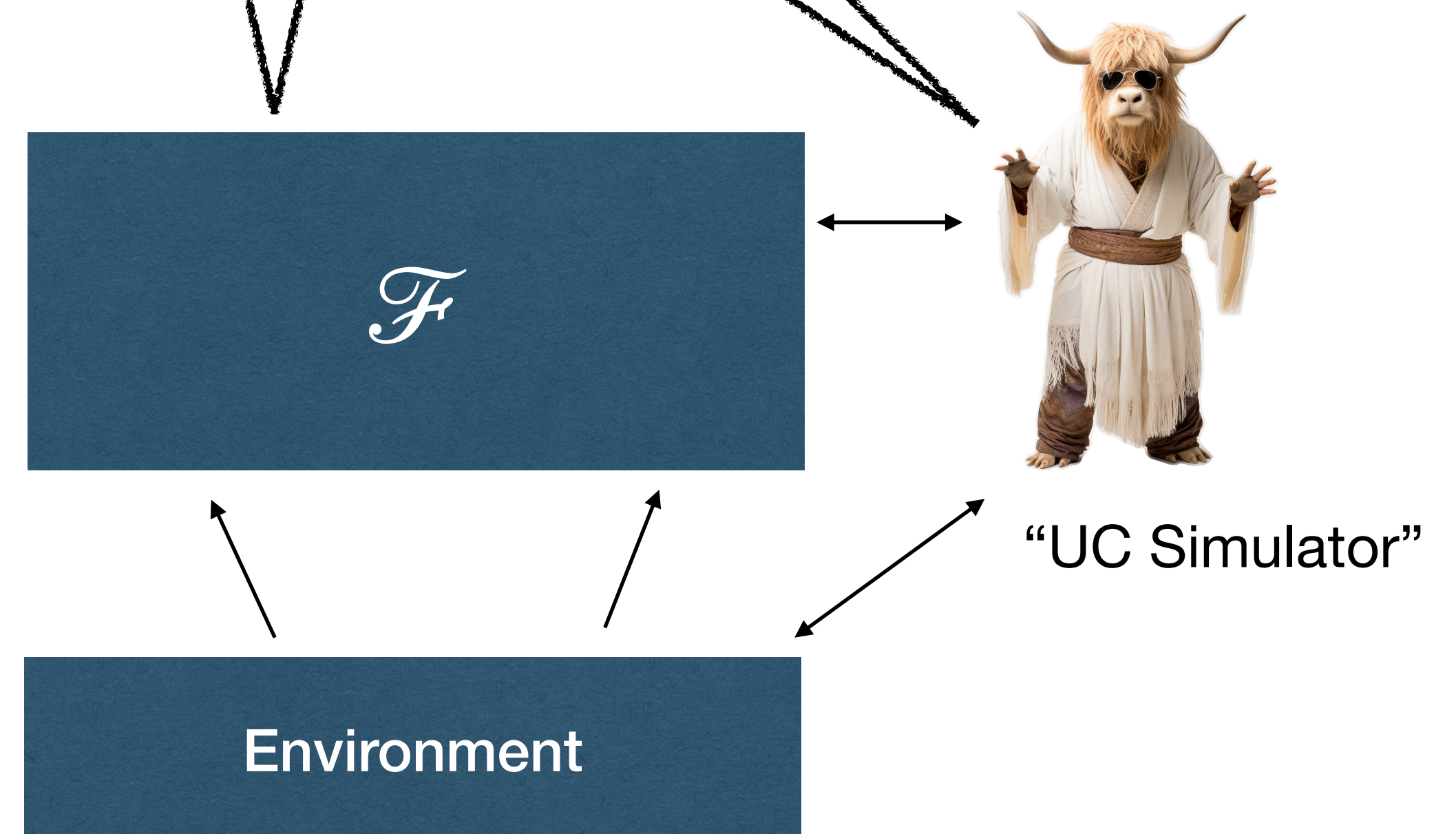
UC modeling

Real world



Ideal world

1. Real protocol *simulated* (using leakage).
2. Adversarial inputs to $\mathcal{F}$ extracted from protocol





Find the Billion Dollar Attack

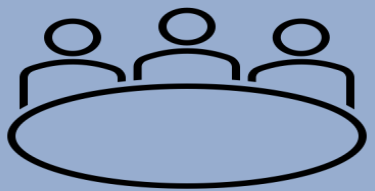
Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



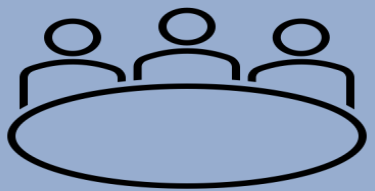
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



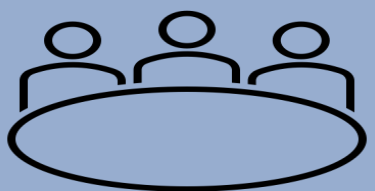
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



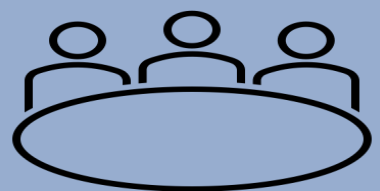
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



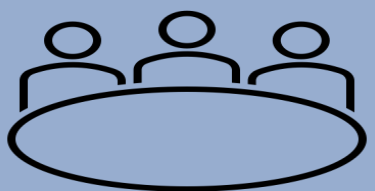
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $R(U_s) = \perp$ or $v < 0$. Else, retrieve $R(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $V(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $V(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $V(\text{ST.idn}) = (U_s, U_r, v)$, and $R(U_s) = B_s$. Set $R(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



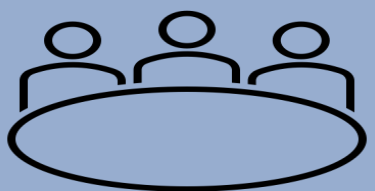
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



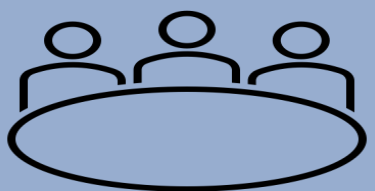
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



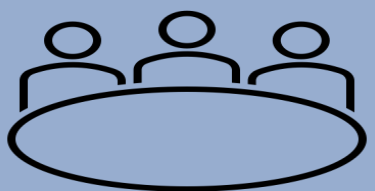
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



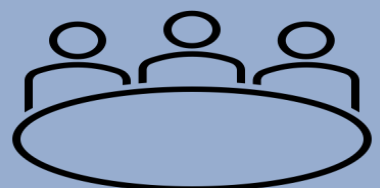
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



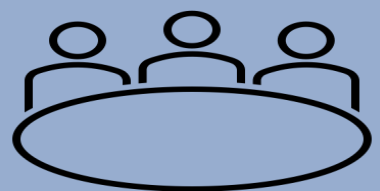
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $R(U_s) = \perp$ or $v < 0$. Else, retrieve $R(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $V(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $V(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $V(\text{ST.idn}) = (U_s, U_r, v)$, and $R(U_s) = B_s$. Set $R(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



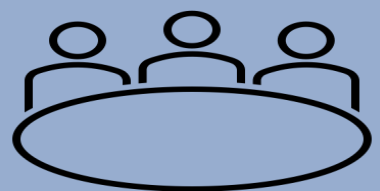
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $R(U_s) = \perp$ or $v < 0$. Else, retrieve $R(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $V(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $V(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $V(\text{ST.idn}) = (U_s, U_r, v)$, and $R(U_s) = B_s$. Set $R(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



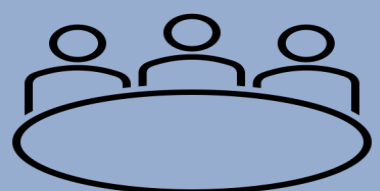
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $R(U_s) = \perp$ or $v < 0$. Else, retrieve $R(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $V(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $V(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $V(\text{ST.idn}) = (U_s, U_r, v)$, and $R(U_s) = B_s$. Set $R(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



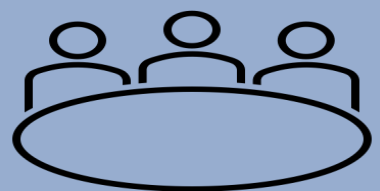
Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $R(U_s) = \perp$ or $v < 0$. Else, retrieve $R(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $V(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $V(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $V(\text{ST.idn}) = (U_s, U_r, v)$, and $R(U_s) = B_s$. Set $R(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.



Are there any ambiguities in the translation?

$P.\text{Transfer}(P', v):$

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. $B \leftarrow \text{Balance}[P]$
6. If $\text{Transaction}[\phi] \neq \perp \vee B < v$ then return $\perp$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

Find the Billion Dollar Attack

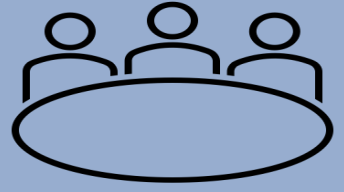
Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. $B \leftarrow \text{Balance}[P]$
6. If $\text{Transaction}[\phi] \neq \perp \vee B < v$ then return $\perp$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ



Which line of code prevents the attack?

Find the Billion Dollar Attack

Stablecoin Transfer. (i) Upon receiving a message $(\text{Gen.ST}, \text{sid}, U_r, v)$ from some party U_s : Ignore if $\mathbb{R}(U_s) = \perp$ or $v < 0$. Else, retrieve $\mathbb{R}(U_s) = B_s$. Ignore if $B_s < v$. Else, generate a new ST.idn and set $\mathbb{V}(\text{ST.idn}) \leftarrow (U_s, U_r, v)$. Output $(\text{Gen.ST}, \text{sid}, \text{ST.idn})$ to $\mathcal{A}$. (ii) Upon receiving $(\text{Gen.ST.Ok}, \text{sid}, \text{ST.idn}, \phi)$ from $\mathcal{A}$: Ignore if $\mathbb{V}(\text{ST.idn}) = \perp$ or there already exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. Else, retrieve $\mathbb{V}(\text{ST.idn}) = (U_s, U_r, v)$, and $\mathbb{R}(U_s) = B_s$. Set $\mathbb{R}(U_s) \leftarrow (B_s - v)$. Record $(\phi, U_s, U_r, v, 0)$. Output $(\text{Gen.ST.End}, \text{sid}, U_r, v, \phi)$ to U_s via private-delayed output.

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. If $\text{Transaction}[\phi] \neq \perp$ then return $\perp$
6. $B \leftarrow \text{Balance}[P]$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

$P.\text{Transfer}(P', v)$:

1. If $\text{Balance}[P] = \perp \vee v < 0 = \perp$ then return $\perp$
2. $B \leftarrow \text{Balance}[P]$
3. If $B < v$ then return $\perp$
4. $\phi \leftarrow \blacktriangleright \mathcal{S}.\text{Execute-Transaction}$
5. $B \leftarrow \text{Balance}[P]$
6. If $\text{Transaction}[\phi] \neq \perp \vee B < v$ then return $\perp$
7. $\text{Balance}[P] \leftarrow B - v$; $\text{Transaction}[\phi] \leftarrow (P, P', v, 0)$
8. $\blacktriangleright \mathcal{S}.\text{Delay-Output}$
9. return ϕ

zkSNARKs

Modeling security



zkSNARKs



$\text{Prove}(x, w) \rightarrow \pi$ // Computes a short proof π attesting to $(x, w) \in R$



$\text{Verify}(x, \pi) \rightarrow b$ // Checks the proof

zkSNARKs



$\text{Prove}(x, w) \rightarrow \pi$ // Computes a short proof π attesting to $(x, w) \in R$



$\text{Verify}(x, \pi) \rightarrow b$ // Checks the proof

! Zero-Knowledge: π does not reveal *any information* about w (other than existence).

zkSNARKs



$\text{Prove}(x, w) \rightarrow \pi$ // Computes a short proof π attesting to $(x, w) \in R$



$\text{Verify}(x, \pi) \rightarrow b$ // Checks the proof

! Zero-Knowledge: π does not reveal *any information* about w (other than existence).

Soundness: If x has no witness w , then prover cannot compute valid proof π .

zkSNARKs



$\text{Prove}(x, w) \rightarrow \pi$ // Computes a short proof π attesting to $(x, w) \in R$



$\text{Verify}(x, \pi) \rightarrow b$ // Checks the proof

! Zero-Knowledge: π does not reveal *any information* about w (other than existence).

Soundness: If x has no witness w , then prover cannot compute valid proof π .

! Proof of knowledge: In order to compute valid π , prover must *know* w .

zkSNARKs



$\text{Prove}(x, w) \rightarrow \pi$ // Computes a short proof π attesting to $(x, w) \in R$



$\text{Verify}(x, \pi) \rightarrow b$ // Checks the proof

! Zero-Knowledge: π does not reveal *any information* about w (other than existence).

Soundness: If x has no witness w , then prover cannot compute valid proof π .

! Proof of knowledge: In order to compute valid π , prover must *know* w .

Succinctness: Proofs π are short and Verify algorithm is efficient.



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .

Proof of knowledge

In order to compute valid
 π , prover must *know* w .



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .

Proof of knowledge

In order to compute valid
 π , prover must *know* w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

No ppt $\mathcal{A}$ can distinguish
 (x, π) where $\pi \leftarrow \text{Prove}(x, w)$ //honest
from
 (x, π) where $\pi \leftarrow \mathcal{S}(x)$ //simulated



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

No ppt $\mathcal{A}$ can distinguish
 (x, π) where $\pi \leftarrow \text{Prove}(x, w)$ //honest
from
 (x, π) where $\pi \leftarrow \mathcal{S}(x)$ //simulated

Proof of knowledge

In order to compute valid
 π , prover must *know* w .



Efficient Extractor

$\mathcal{E}(x, \pi) \rightarrow w$

No ppt $\mathcal{A}$ can output x, π such that
 $\text{Verify}(x, \pi) = 1$ //valid proof
and
 $(x, \mathcal{E}(x, \pi)) \notin \mathbf{R}$ //extractor fails



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

No ppt $\mathcal{A}$ can distinguish
 (x, π) where $\pi \leftarrow \text{Prove}(x, w)$ //honest
from
 (x, π) where $\pi \leftarrow \mathcal{S}(x)$ //simulated

Contradiction

Proof of knowledge

In order to compute valid
 π , prover must *know* w .



Efficient Extractor

$\mathcal{E}(x, \pi) \rightarrow w$

No ppt $\mathcal{A}$ can output x, π such that
 $\text{Verify}(x, \pi) = 1$ //valid proof
and
 $(x, \mathcal{E}(x, \pi)) \notin \mathbf{R}$ //extractor fails



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

Malicious prover
can run $\mathcal{S}$
to compute π
without knowing w

Contradiction



Efficient Extractor

$\mathcal{E}(x, \pi) \rightarrow w$

No ppt $\mathcal{A}$ can distinguish
 (x, π) where $\pi \leftarrow \text{Prove}(x, w)$ //honest
from

(x, π) where $\pi \leftarrow \mathcal{S}(x)$ //simulated

No ppt $\mathcal{A}$ can output x, π such that
 $\text{Verify}(x, \pi) = 1$ //valid proof
and

$(x, \mathcal{E}(x, \pi)) \notin \mathbf{R}$ //extractor fails



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

Malicious prover
can run $\mathcal{S}$
to compute π
without knowing w

Contradiction



Efficient Extractor

$\mathcal{E}(x, \pi) \rightarrow w$

Malicious verifier
can run $\mathcal{E}$
to learn
full information on w

No ppt $\mathcal{A}$ can distinguish
 (x, π) where $\pi \leftarrow \text{Prove}(x, w)$ //honest
from
 (x, π) where $\pi \leftarrow \mathcal{S}(x)$ //simulated

No ppt $\mathcal{A}$ can output x, π such that
 $\text{Verify}(x, \pi) = 1$ //valid proof
and
 $(x, \mathcal{E}(x, \pi)) \notin \mathbf{R}$ //extractor fails



$\text{Prove}(x, w) \rightarrow \pi$

zkSNARKs

$\text{Verify}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S}(x) \rightarrow \pi$

Malicious prover
can run $\mathcal{S}$
to compute π
without knowing w



Efficient Extractor

$\mathcal{E}(x, \pi) \rightarrow w$

Malicious verifier
can run $\mathcal{E}$
to learn
full information on w

Simulator and extractor need a
superpower

that malicious provers/verifiers don't have.



Simulator and extractor need a **superpower**

that malicious provers/verifiers don't have.

 Strategy



Real life



Superpower
of Simulator/Extractor



Malicious
prover/verifier



Simulator and extractor need a superpower

that malicious provers/verifiers don't have.

 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs (incl. randomness)	Cannot re-run party



Simulator and extractor need a superpower

that malicious provers/verifiers don't have.



 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs (incl. randomness)	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor

Simulator and extractor need a superpower

that malicious provers/verifiers don't have.



 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs (incl. randomness)	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values

Simulator and extractor need a superpower

that malicious provers/verifiers don't have.



Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs (incl. randomness)	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations

Groth16





CRS: group elements $u_i, \alpha, \beta, \delta$

Witness: wire values $a_i \in \mathbb{Z}_p$



Groth16





CRS: group elements $u_i, \alpha, \beta, \delta$

Witness: wire values $a_i \in \mathbb{Z}_p$



Groth16



$$A = \sum_{i=0}^m a_i u_i + \alpha + r \delta$$
$$B = \sum_{i=0}^m a_i v_i + \beta + r' \delta$$
$$C = \dots$$



[Check some pairing equations on A,B,C]



CRS: group elements $u_i, \alpha, \beta, \delta$

Witness: wire values $a_i \in \mathbb{Z}_p$



Groth16

Extractor



$$A = \sum_{i=0}^m a_i u_i + \alpha + r\delta$$
$$B = \sum_{i=0}^m a_i v_i + \beta + r'\delta$$
$$C = \dots$$



[Check some pairing equations on A,B,C]



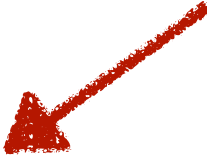
Prove^o_(x, w) → π

zkSNARKs

Verify^o_(x, π) → b





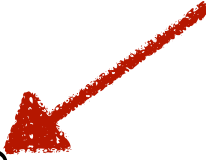
Prove^{} $\mathcal{O}(x, w) \rightarrow \pi$

zkSNARKs



with superpowers:

control over resource $\mathcal{O}$

Verify^{} $\mathcal{O}(x, \pi) \rightarrow b$





$\text{Prove}^{\mathcal{O}}(x, w) \rightarrow \pi$

zkSNARKs



with superpowers:

control over resource $\mathcal{O}$

$\text{Verify}^{\mathcal{O}}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S} . \mathcal{O} \leftarrow$

$\mathcal{S} . \text{Sim}(x) \rightarrow \pi$

Indistinguishable for ppt $\mathcal{A}$:

$\mathcal{A}^{\mathcal{O}}, (x, w) \mapsto \text{Prove}^{\mathcal{O}}(x, w)$ //honest $\mathcal{O}$ /proofs

from

$\mathcal{A}^{\mathcal{S} . \mathcal{O}}, R \ni (x, w) \mapsto \mathcal{S} . \text{Sim}(x)$ //simulated $\mathcal{O}$ /proofs



$\text{Prove}^{\mathcal{O}}(x, w) \rightarrow \pi$

zkSNARKs



with superpowers:

control over resource $\mathcal{O}$

$\text{Verify}^{\mathcal{O}}(x, \pi) \rightarrow b$



Zero-Knowledge

π does not reveal
any information about w .



Efficient Simulator

$\mathcal{S} . \mathcal{O} \leftarrow$

$\mathcal{S} . \text{Sim}(x) \rightarrow \pi$

Indistinguishable for ppt $\mathcal{A}$:

$\mathcal{A}^{\mathcal{O}}, (x, w) \mapsto \text{Prove}^{\mathcal{O}}(x, w)$ //honest $\mathcal{O}$ /proofs

from

$\mathcal{A}^{\mathcal{S} . \mathcal{O}}, \mathbf{R} \ni (x, w) \mapsto \mathcal{S} . \text{Sim}(x)$ //simulated $\mathcal{O}$ /proofs



Efficient Extractor

$\mathcal{E}(x, \pi, st_{\mathcal{S}}) \rightarrow w$

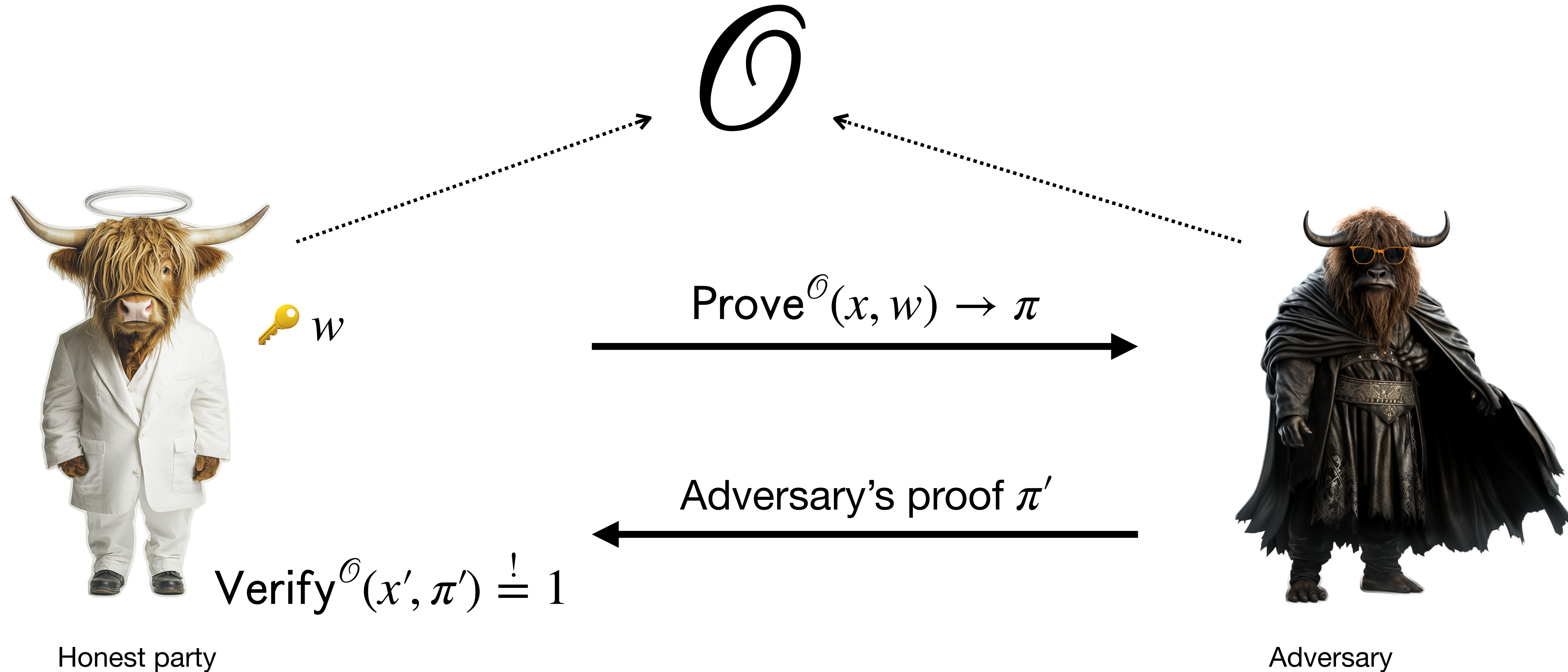
No ppt $\mathcal{A}^{\mathcal{S} . \mathcal{O}}, \mathcal{S} . \text{Sim}$ can output x, π s.t.:

$\text{Verify}^{\mathcal{S} . \mathcal{O}}(x, \pi) = 1$ //valid proof

$\mathcal{A}$ made no query $\mathcal{S} . \text{Sim}(x)$

$(x, \mathcal{E}(x, \pi, st_{\mathcal{S}})) \notin \mathbf{R}$ //extractor fails

Example system involving zkSNARK





Applying the definition

$\mathcal{O}$



w

$\text{Verify}^{\mathcal{O}}(x', \pi') \stackrel{!}{=} 1$

Honest party

$\text{Prove}^{\mathcal{O}}(x, w) \rightarrow \pi$

Adversary's proof π'



Adversary



Applying the definition

$\mathcal{O}$



w

$\text{Verify}^{\mathcal{O}}(x', \pi') \stackrel{!}{=} 1$

Honest party

$\text{Prove}^{\mathcal{O}}(x, w) \rightarrow \pi$

Adversary's proof π'



Adversary



Applying the definition

$\mathcal{S}.\mathcal{O}$



$\mathcal{S}.\text{Sim}(x) \rightarrow \pi$

Adversary's proof π'

$\text{Verify}^{\mathcal{S}.\mathcal{O}}(x', \pi') \stackrel{!}{=} 1$

Honest party

Adversary



Applying the definition

$\mathcal{S}.\mathcal{O}$



$\mathcal{S}.\text{Sim}(x) \rightarrow \pi$



Adversary's proof π'



$\text{Verify}^{\mathcal{S}.\mathcal{O}}(x', \pi') \stackrel{!}{=} 1$



Honest party

$\mathcal{E}(x', \pi', st_{\mathcal{S}}) \rightarrow w'$

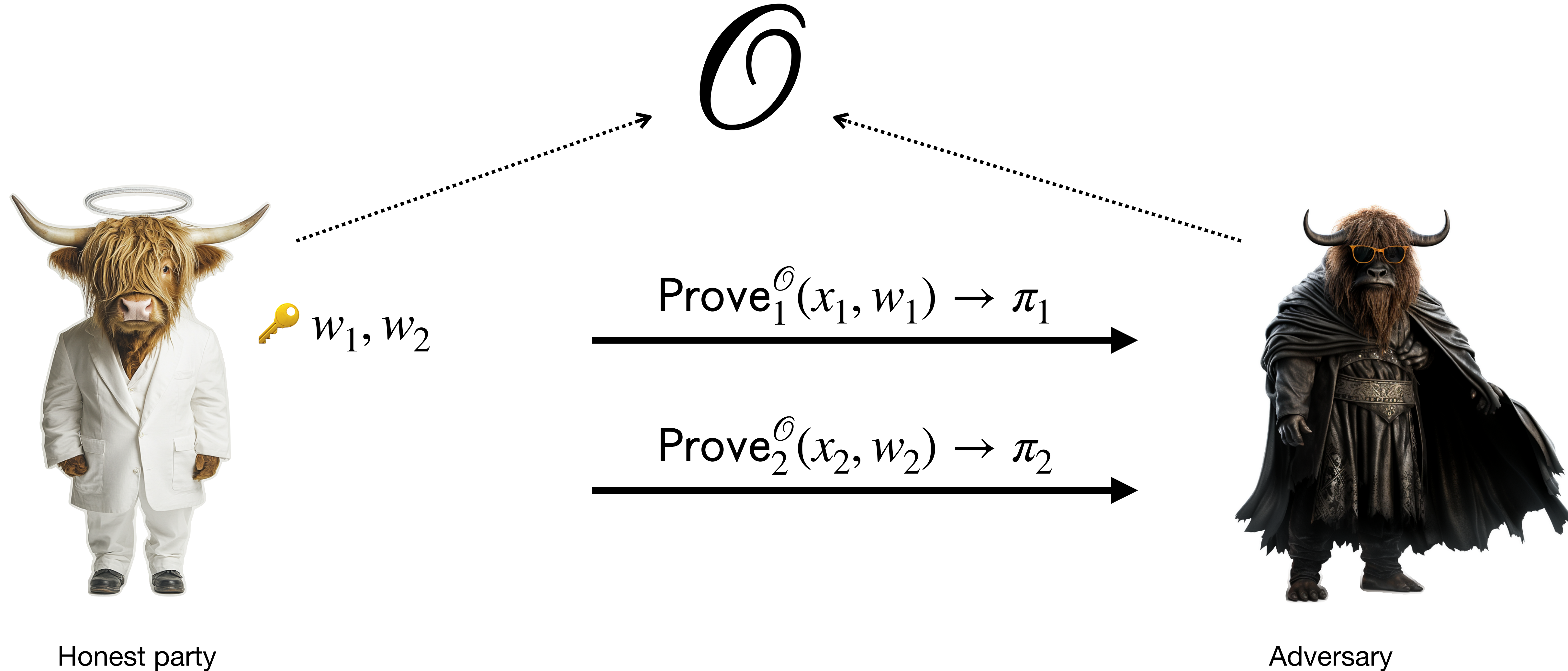


Adversary

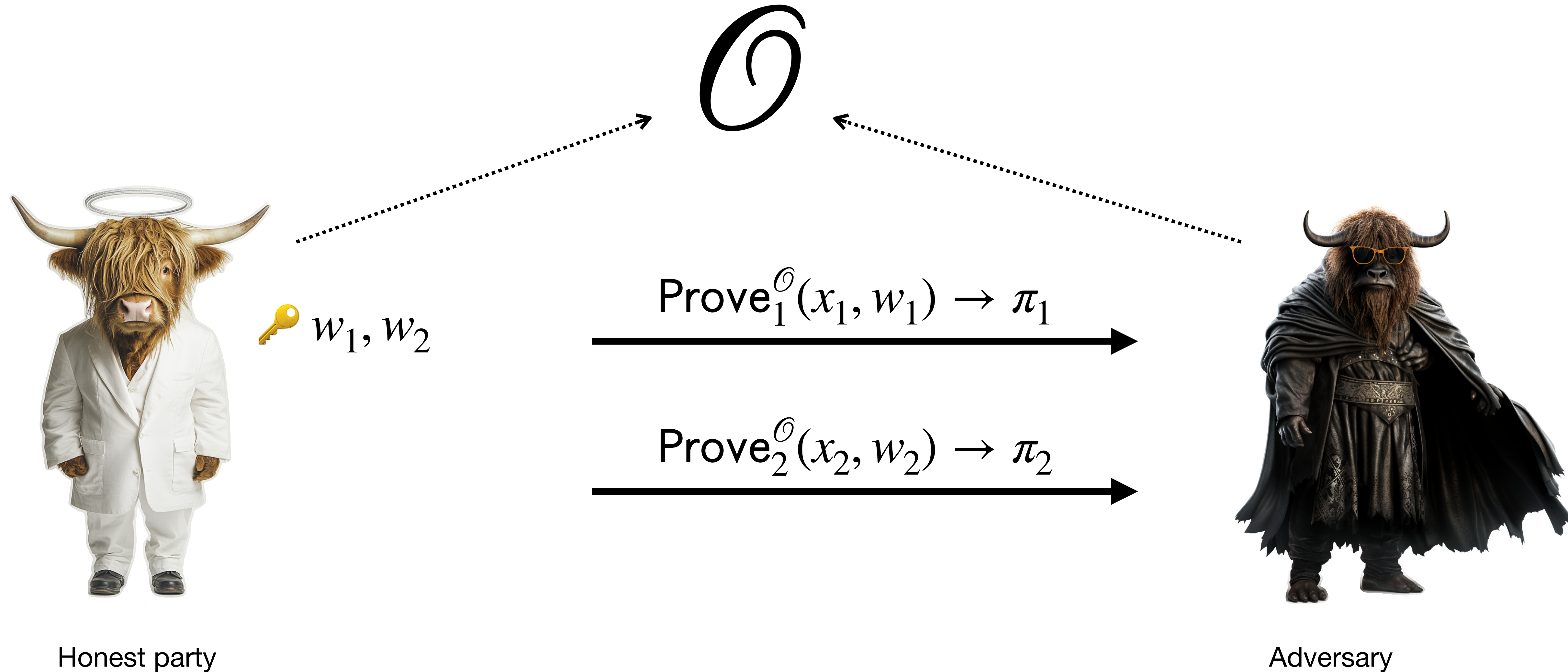
Composition



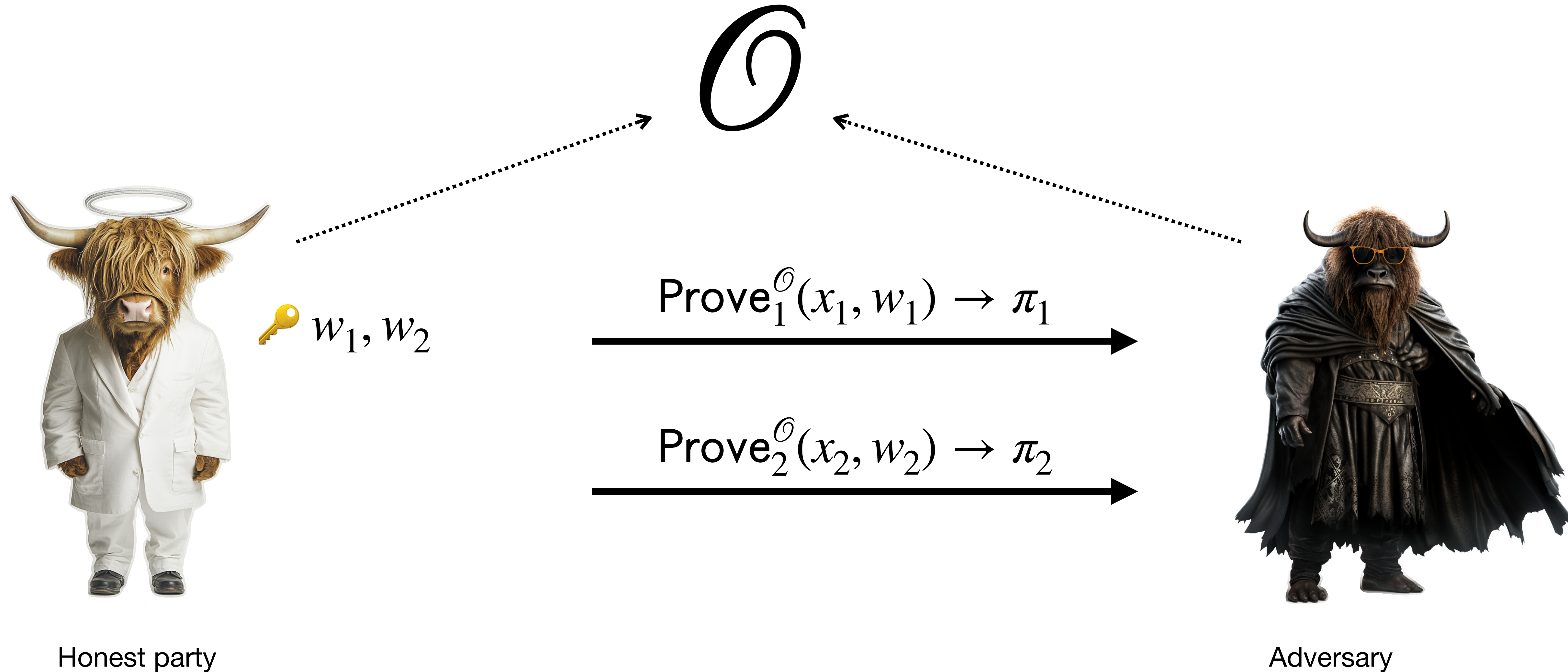
A system with two zkSNARKs



💭 Applying the definition ... twice?



💭 Applying the definition ... twice?



💭 Applying the definition ... twice?

$\mathcal{S}_1 \cdot \mathcal{O}$



Honest party

🔑 w_1, w_2

$\mathcal{S}_1 \cdot \text{Sim}(x_1) \rightarrow \pi_1$

$\text{Prove}_2^{\mathcal{O}}(x_2, w_2) \rightarrow \pi_2$



Adversary

💭 Applying the definition ... twice?

$\mathcal{S}_1 \cdot \mathcal{O}$



Honest party

🔑 ~~w_1, w_2~~

$\mathcal{S}_1 \cdot \text{Sim}(x_1) \rightarrow \pi_1$

$\text{Prove}_2^{\mathcal{O}}(x_2, w_2) \rightarrow \pi_2$



Adversary

💭 Applying the definition ... twice?

$\mathcal{S}_1 \cdot \mathcal{O}$



Honest party

🔑 ~~w_1, w_2~~

$\mathcal{S}_1 \cdot \text{Sim}(x_1) \rightarrow \pi_1$

$\text{Prove}_2^{\mathcal{O}}(x_2, w_2) \rightarrow \pi_2$



Adversary

Cannot replace Prove_2 with $\mathcal{S}_2 \cdot \text{Sim}$:
⚠️ Control of resource $\mathcal{O}$ already taken by $\mathcal{S}_1$

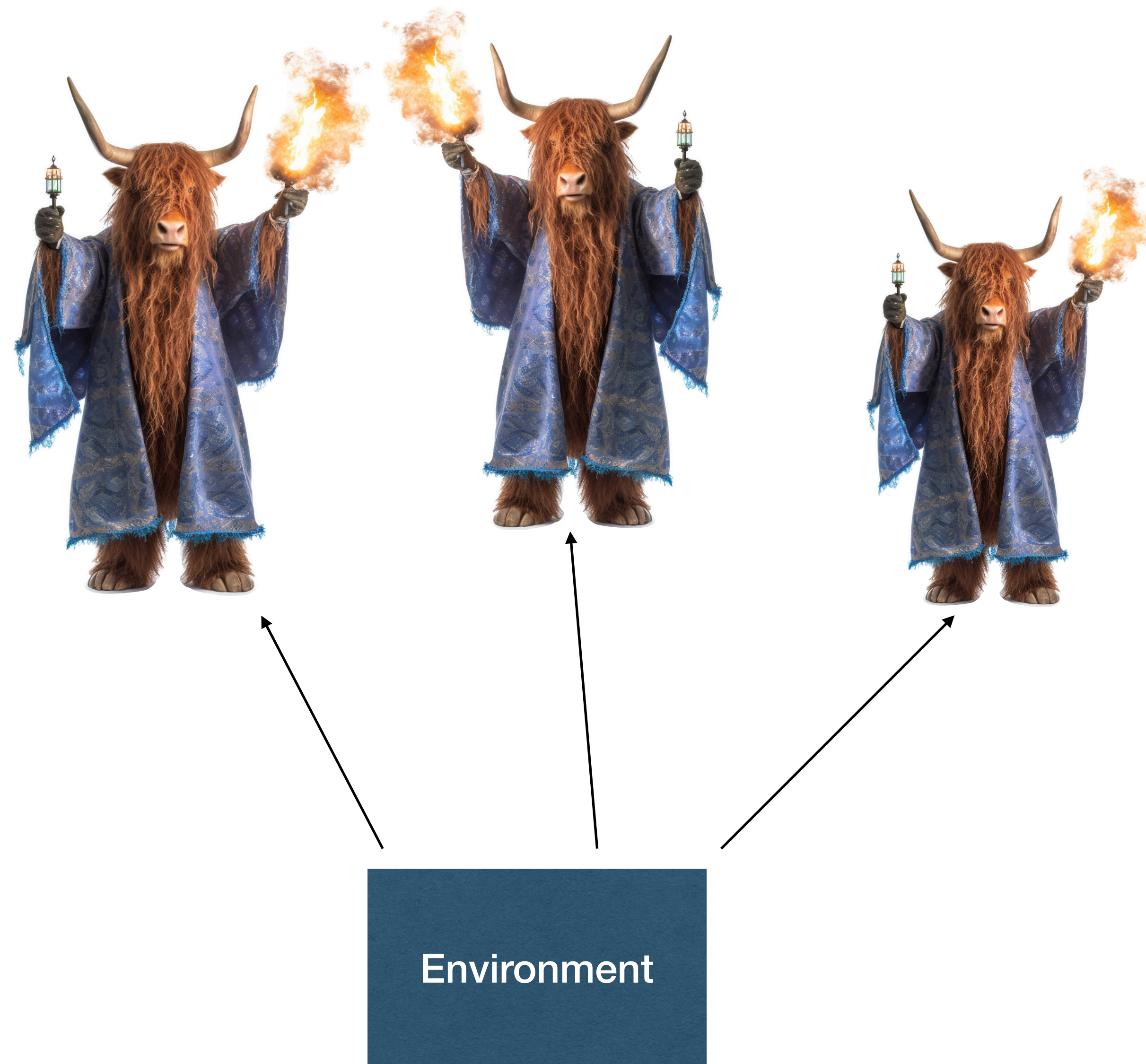
What's the issue?

- Both simulators $\mathcal{S}_1, \mathcal{S}_2$ want **exclusive** control over $\mathcal{O}$
- **No notion** of “sharing” $\mathcal{O}$
 - But: Sharing group/hash function is **very natural** in practice
- An issue of **composability**:
NIZK works **fine by itself**,
but in presence of another NIZK ... 🔥



Universal composability

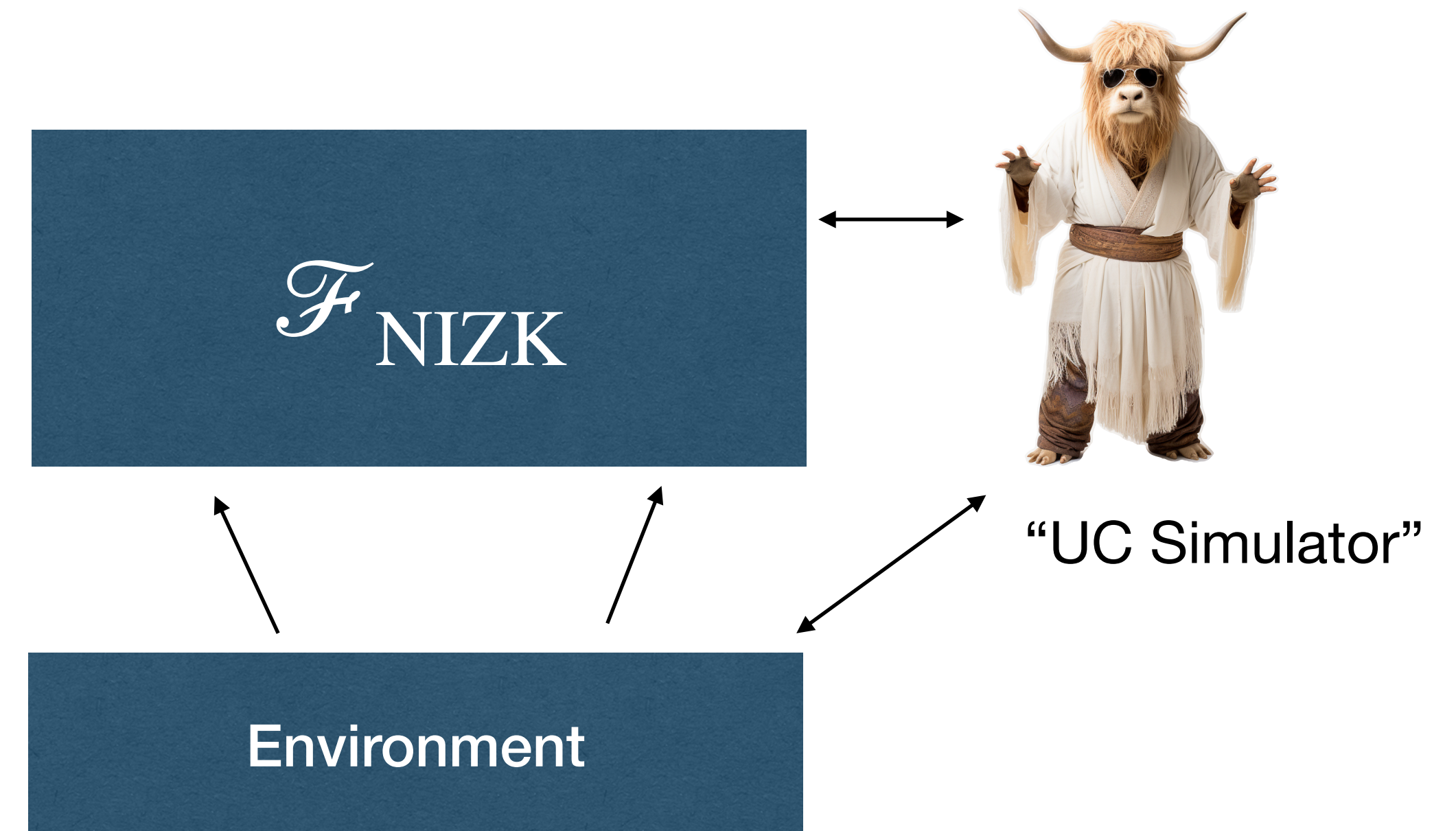
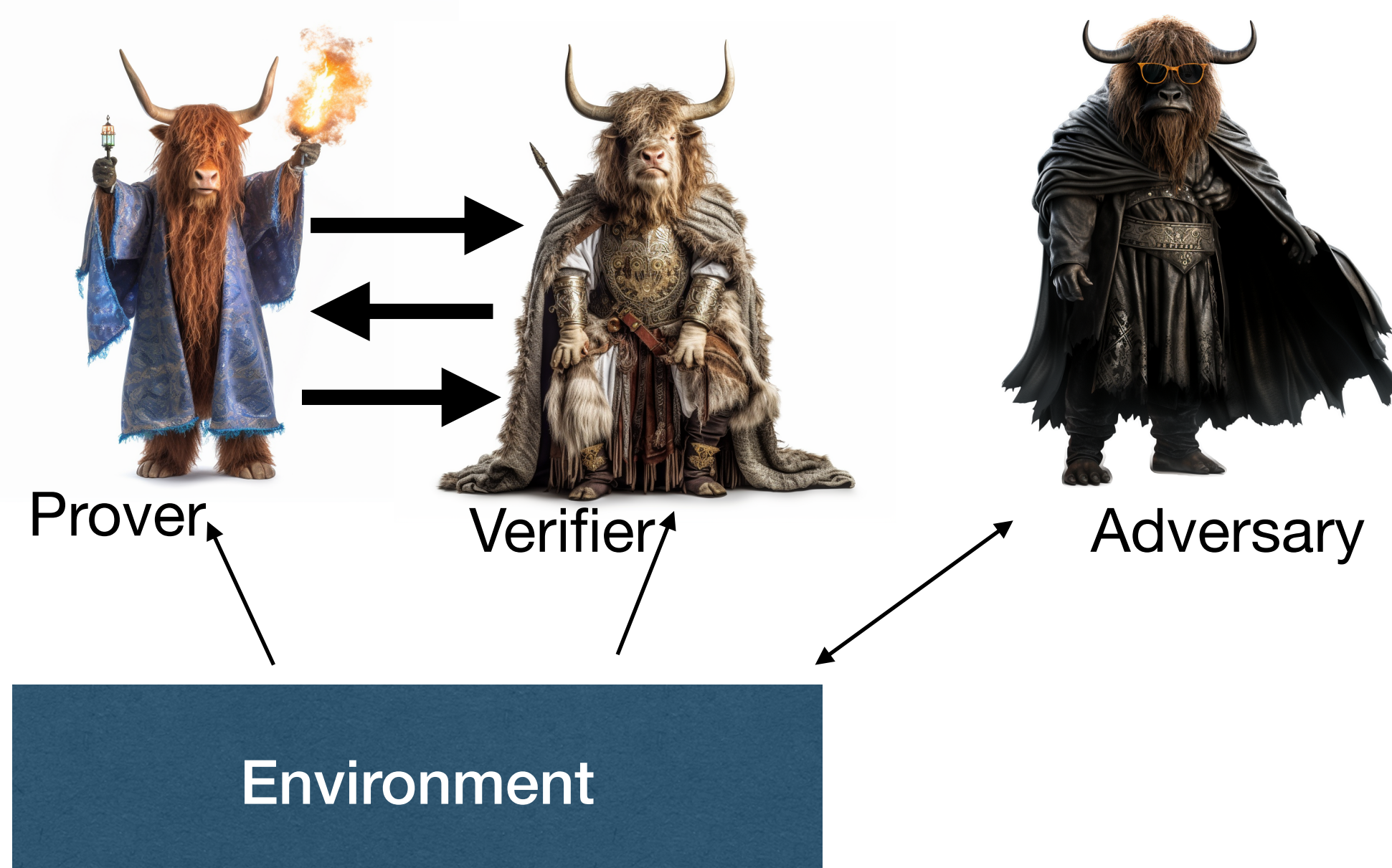
... lets us talk about composition
issues



UC modeling

Real world

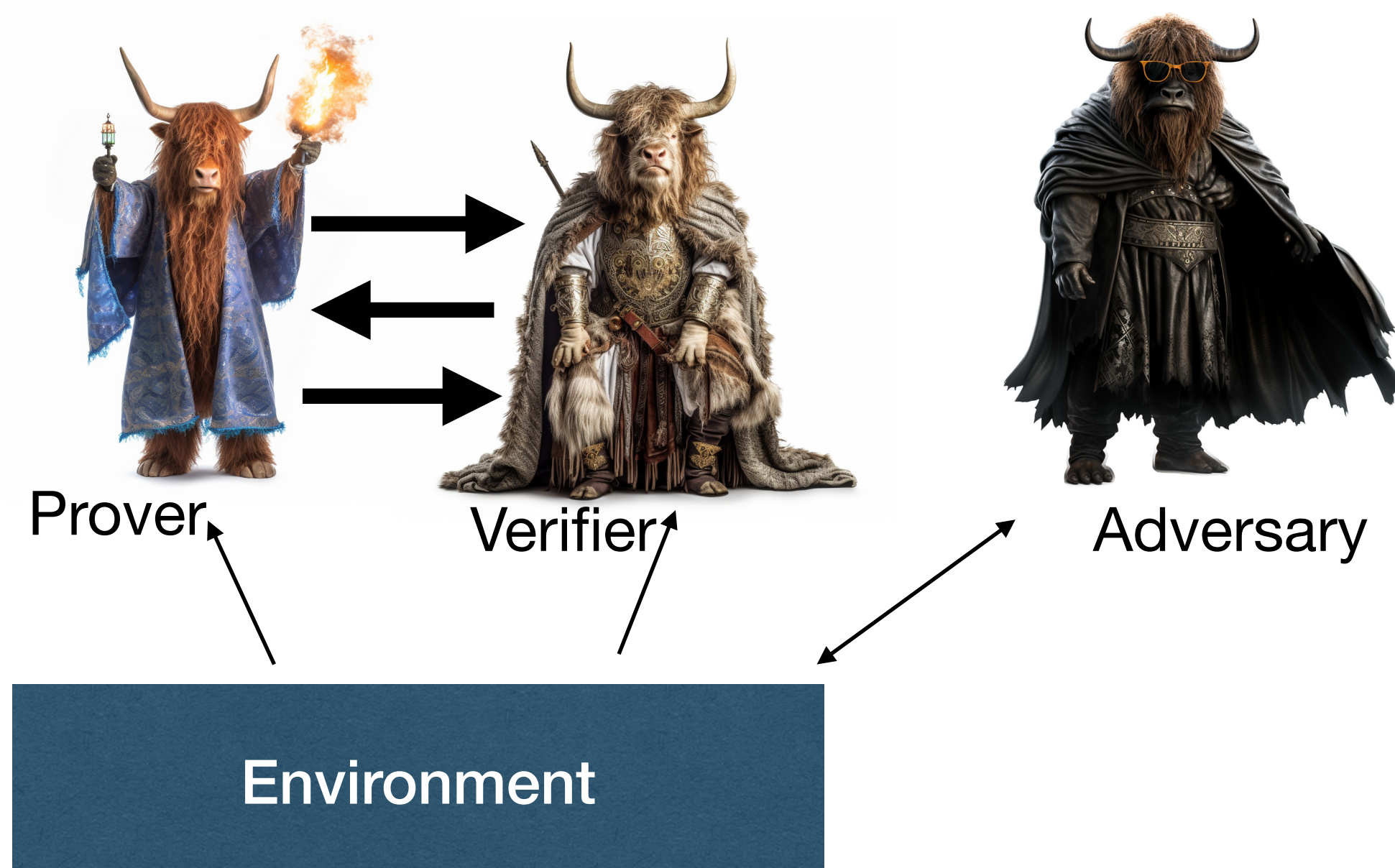
Ideal world



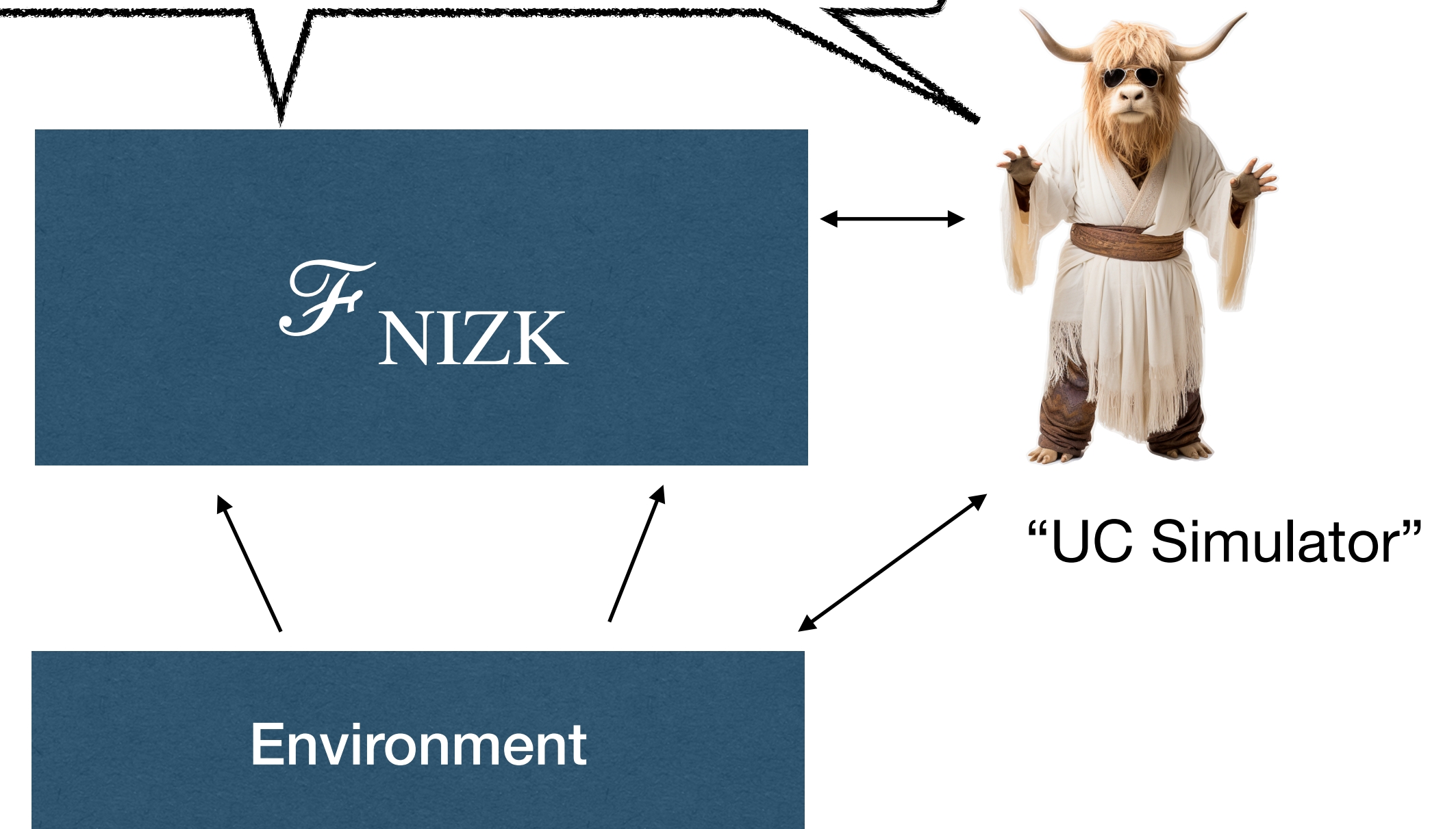
UC modeling

Real world

Ideal world



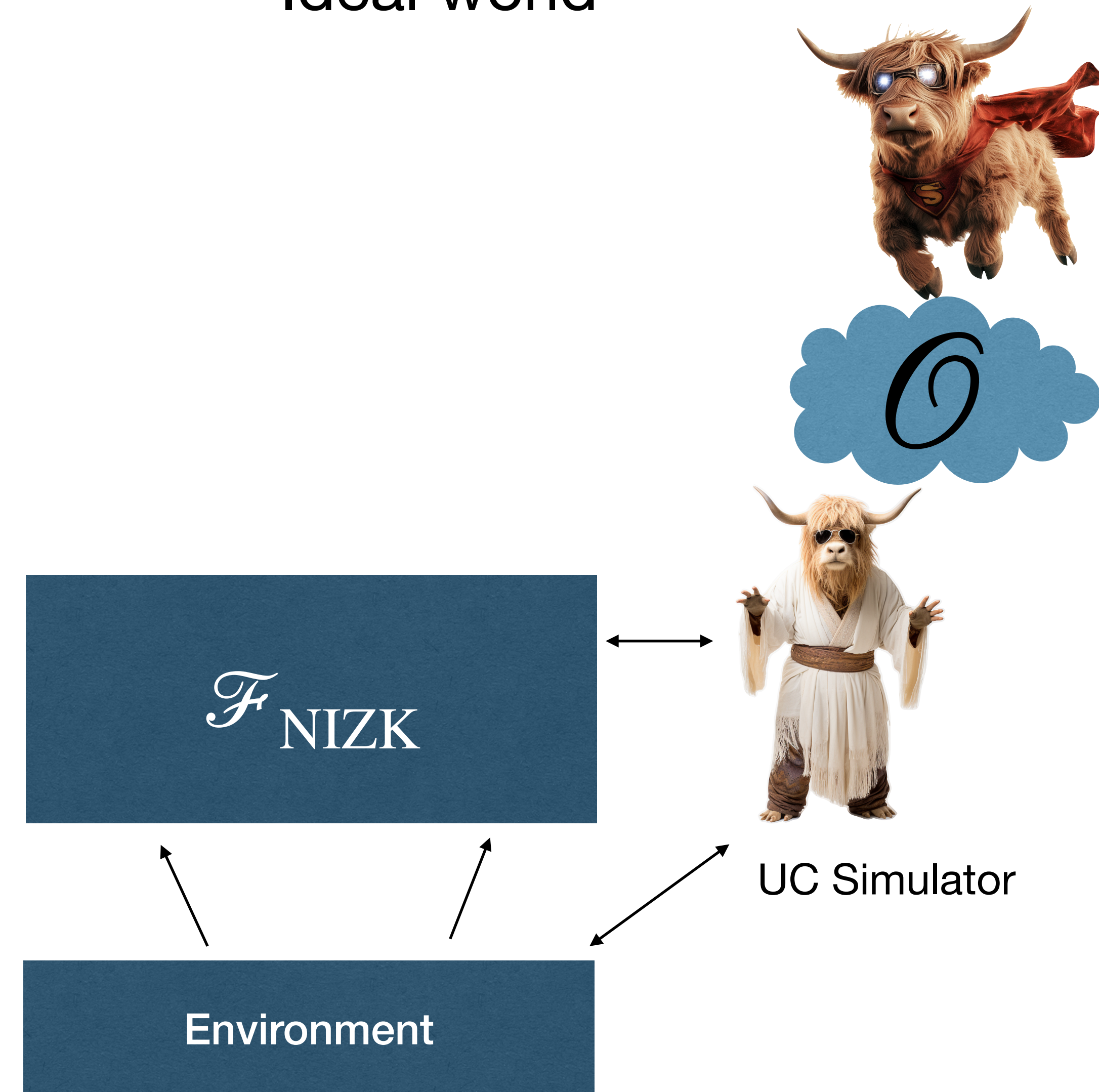
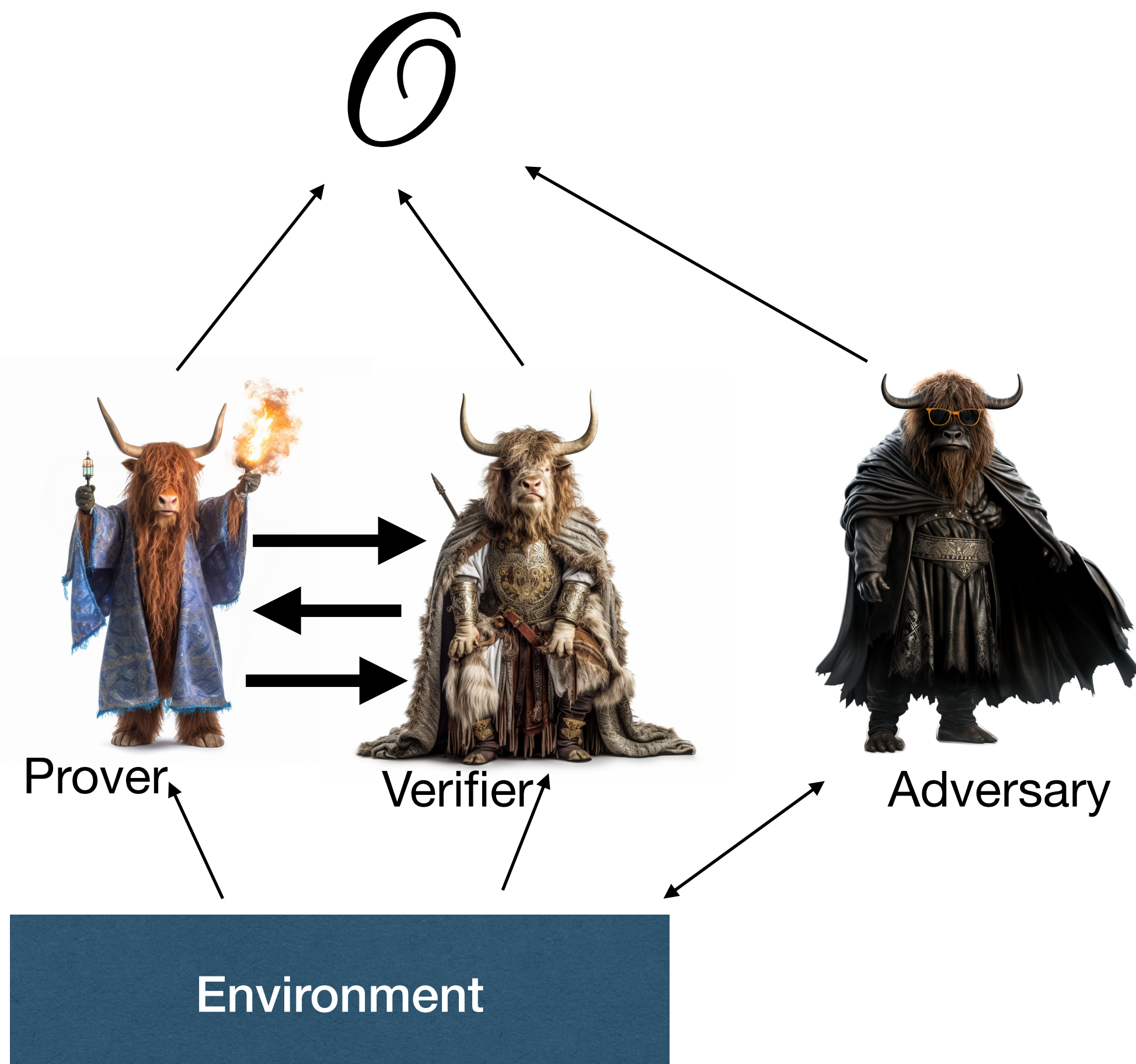
1. Proof are *simulated* (without w).
2. Other proofs are only deemed *valid* if *extractable*.



UC modeling ($\mathcal{O}$ -hybrid)

Real (hybrid) world

Ideal world



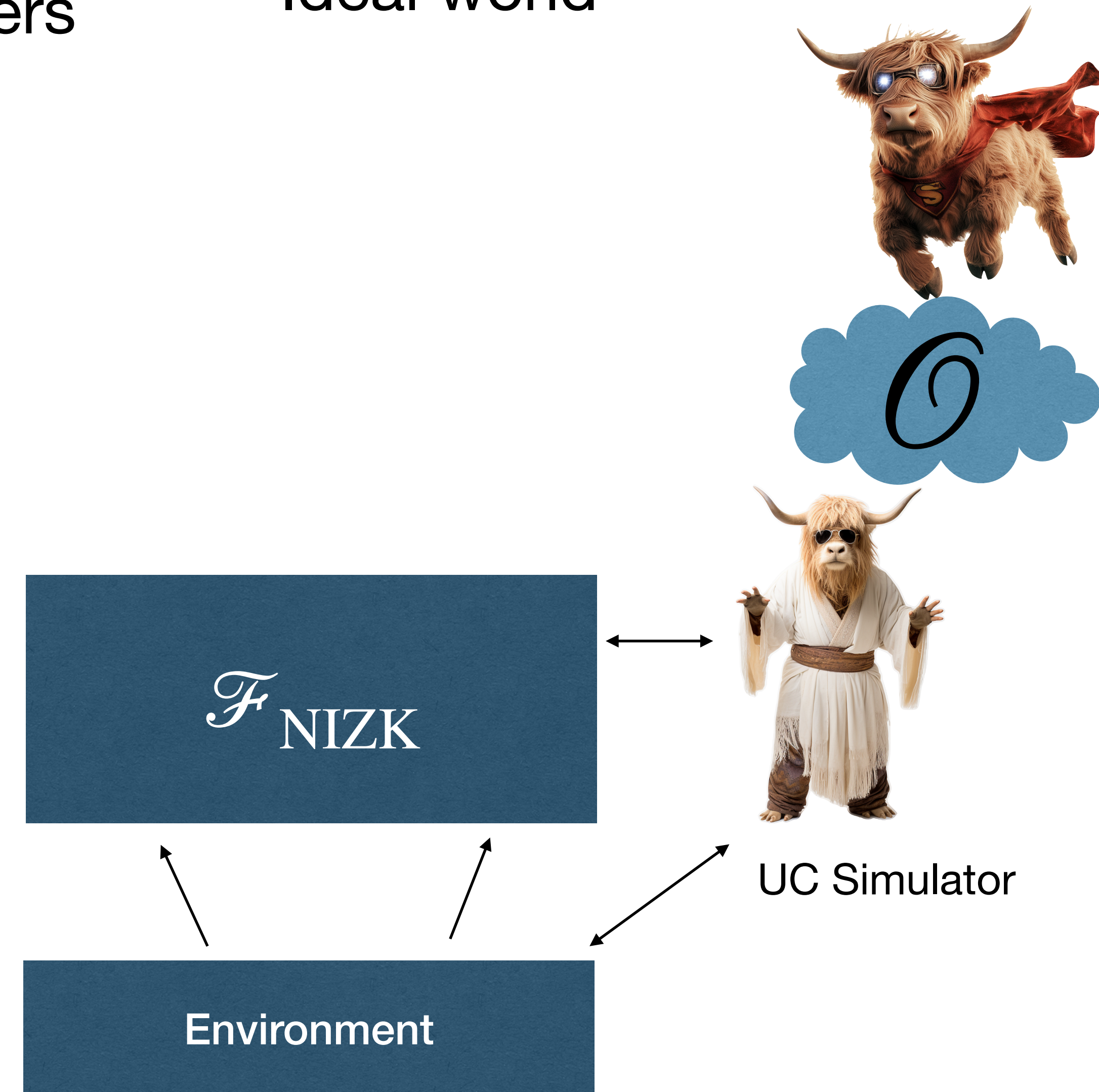
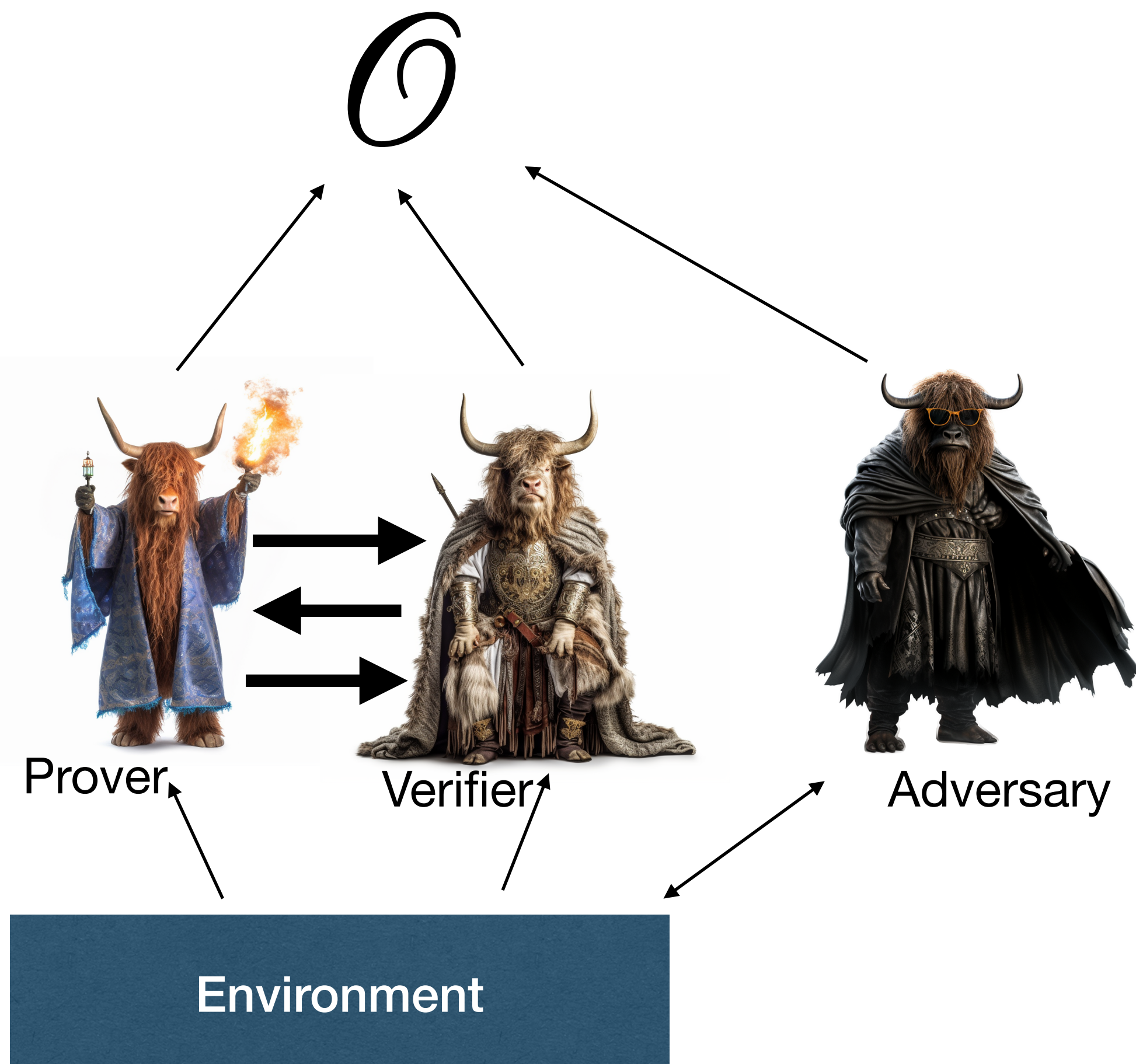
UC modeling ($\mathcal{O}$ -hybrid)

Real (hybrid) world



with superpowers

Ideal world



UC modeling ($\mathcal{O}$ -hybrid)

Real (hybrid) world



with superpowers

Ideal world



Does *not* model sharing of $\mathcal{O}$ with other protocols.

$\mathcal{O}$



Prover



Verifier



Adversary

Environment

$\mathcal{F}_{\text{NIZK}}$

Environment



UC Simulator

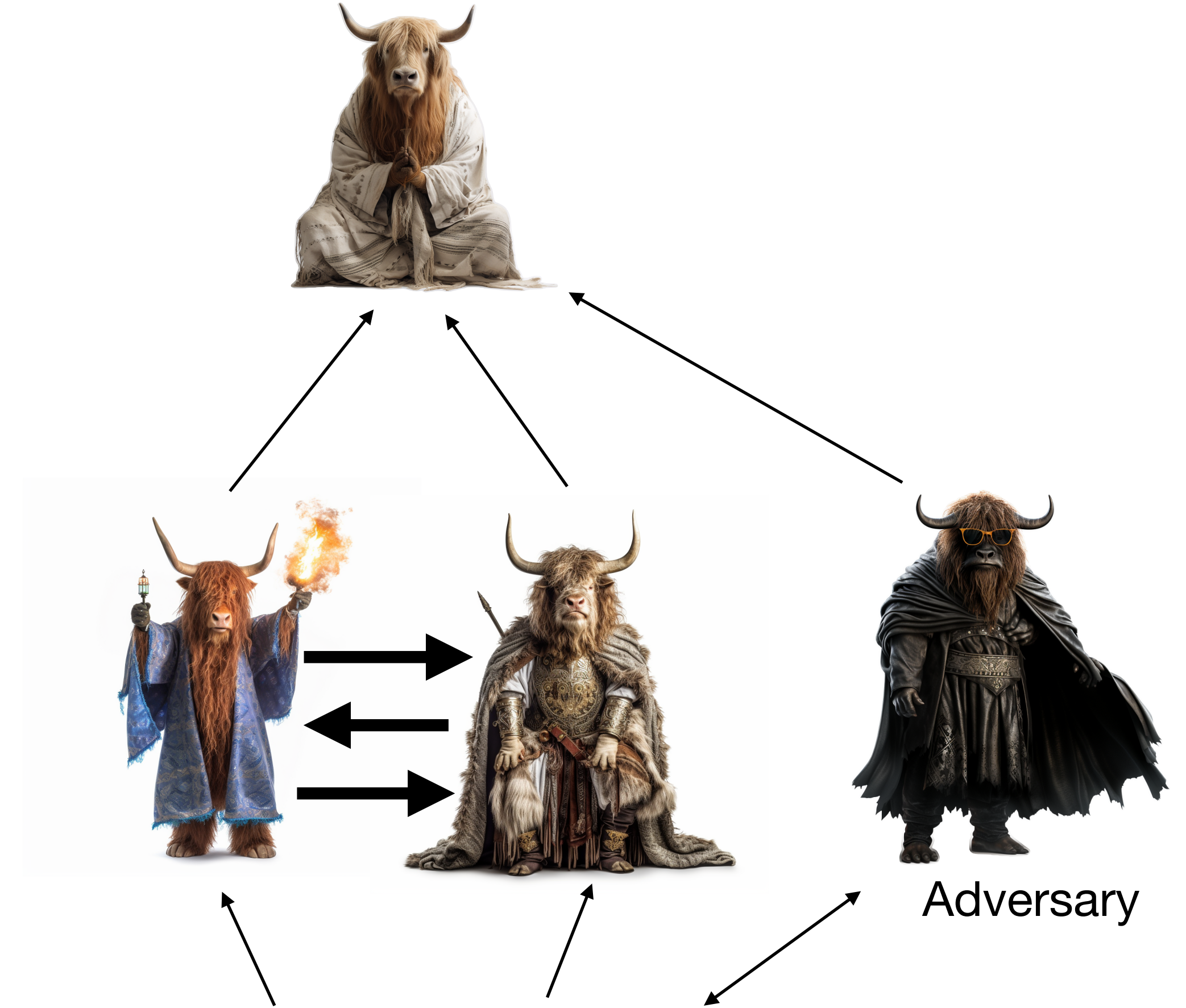
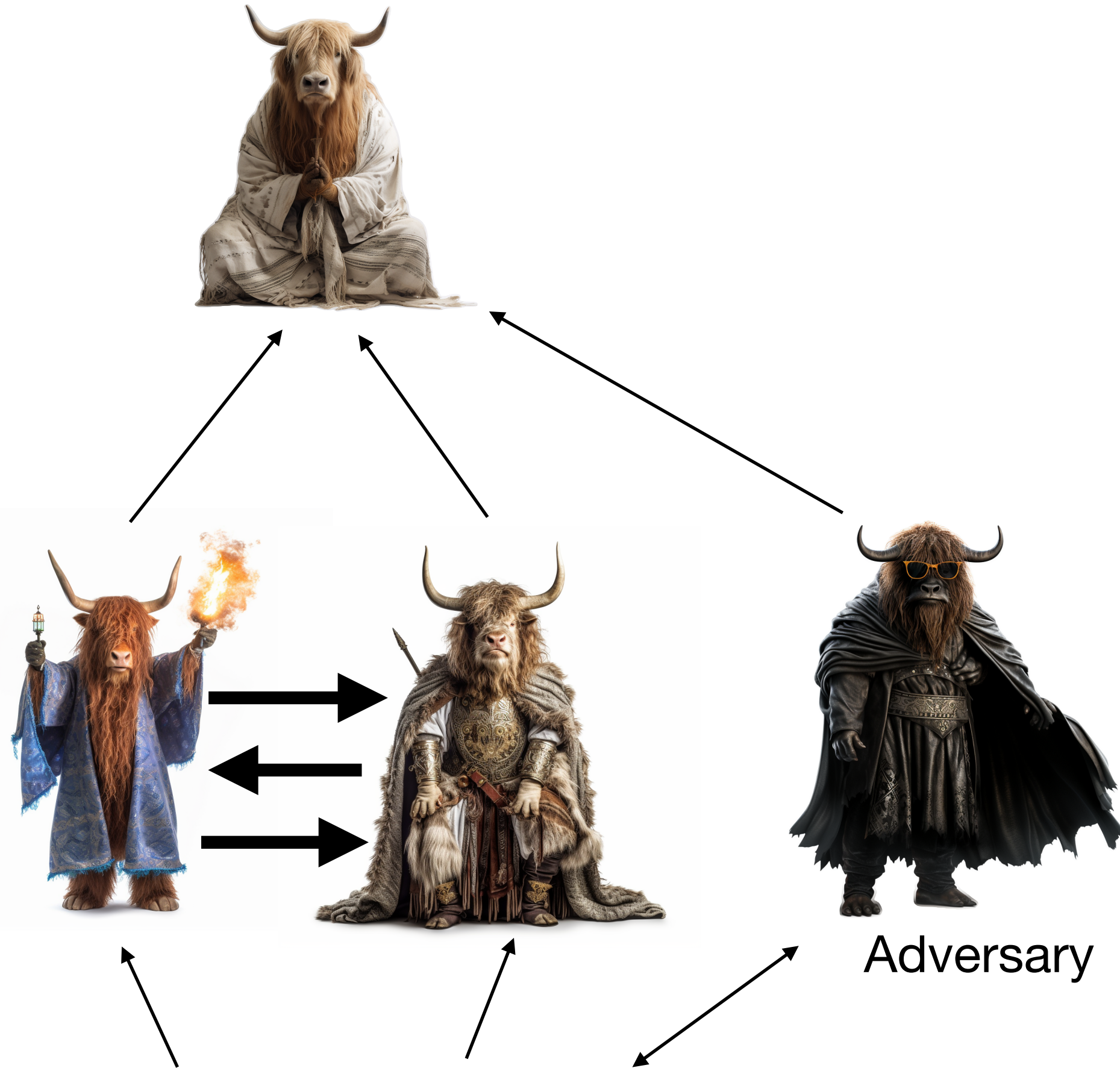


Is this a good model?

Let's have two sessions of the protocol

Random Oracle

Random Oracle



Environment

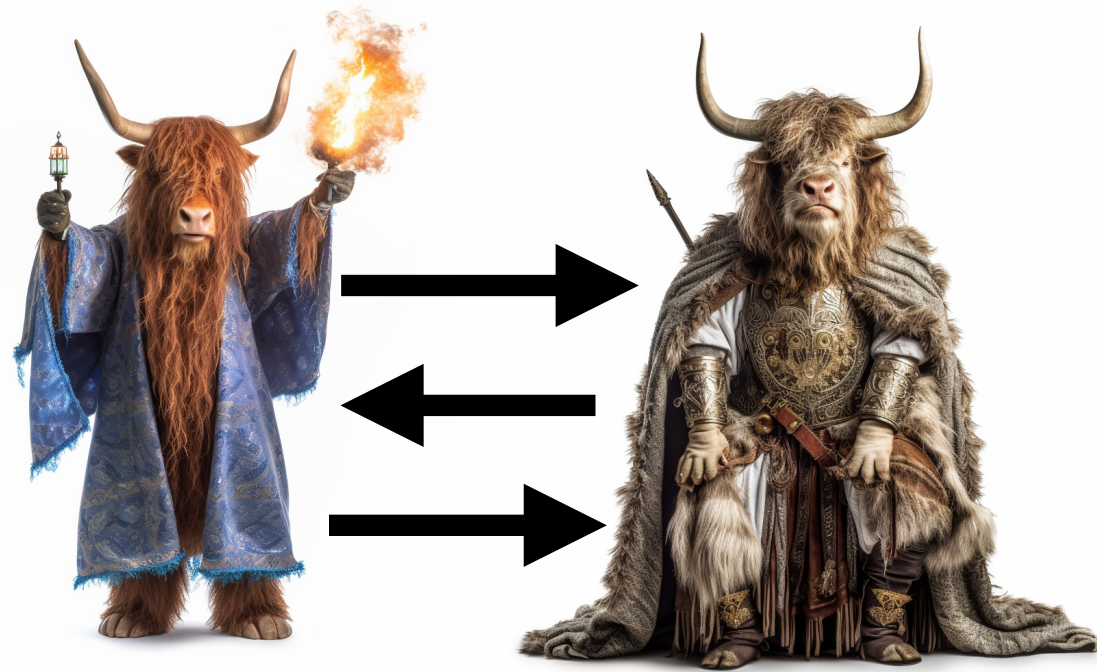
Is this a good model?

Let's have two sessions of the protocol

Random Oracle



Random Oracle



Adversary

What's RO(123) ?



Adversary

What's RO(123) ?

Environment

Is this a good model?

Let's have two sessions of the protocol

Random Oracle

RO(123) = a03ab19b866fc

Random Oracle

What's RO(123) ?

Environment

Is this a good model?

Let's have two sessions of the protocol

Random Oracle



RO(123) = a03ab19b866fc

Random Oracle



RO(123) = 7106b623725f



Adversary

What's RO(123) ?



Adversary

What's RO(123) ?

Environment

Is this a good model?

Let's have two sessions of the protocol

Random Oracle



RO(123) = a03ab19b866fc

Random Oracle



RO(123) = 7106b623725f

This does not model reality!



Adversary

What's RO(123) ?

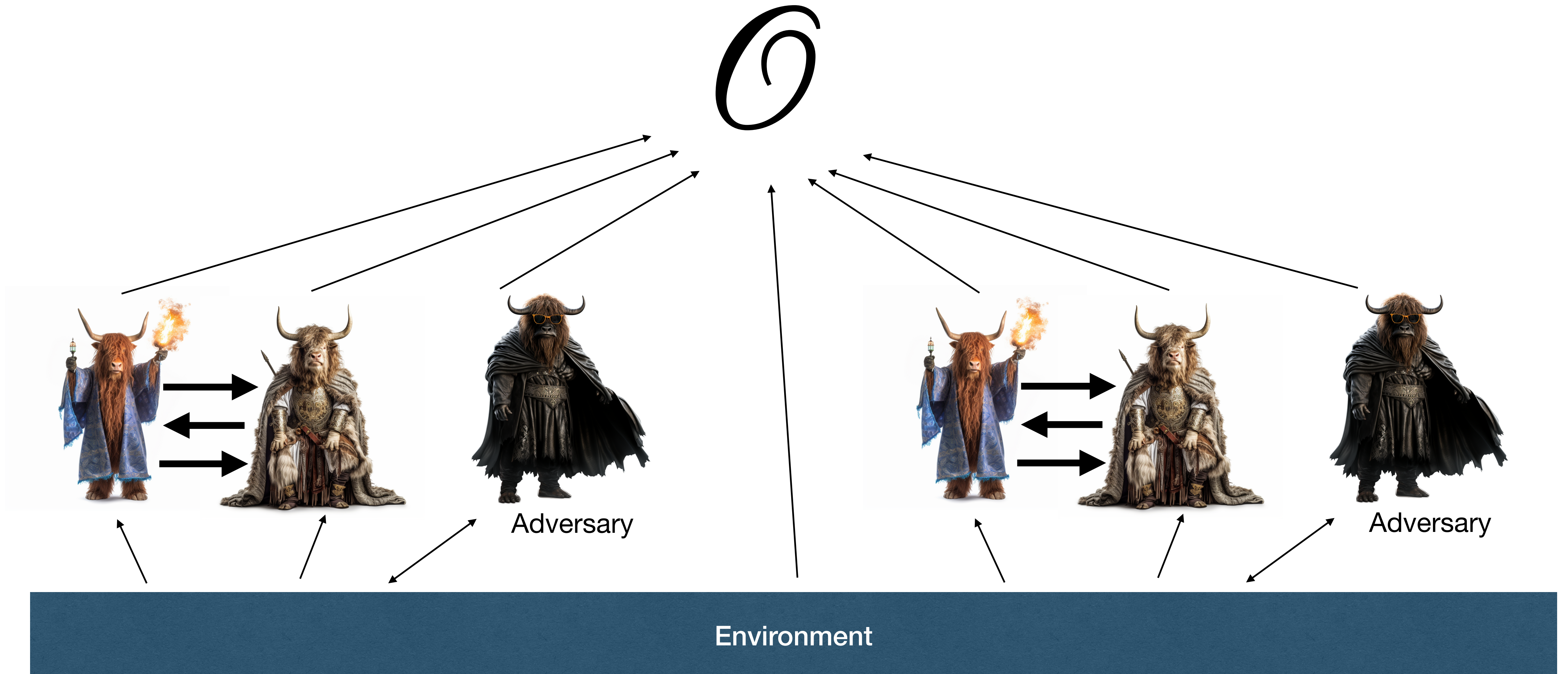


Adversary

What's RO(123) ?

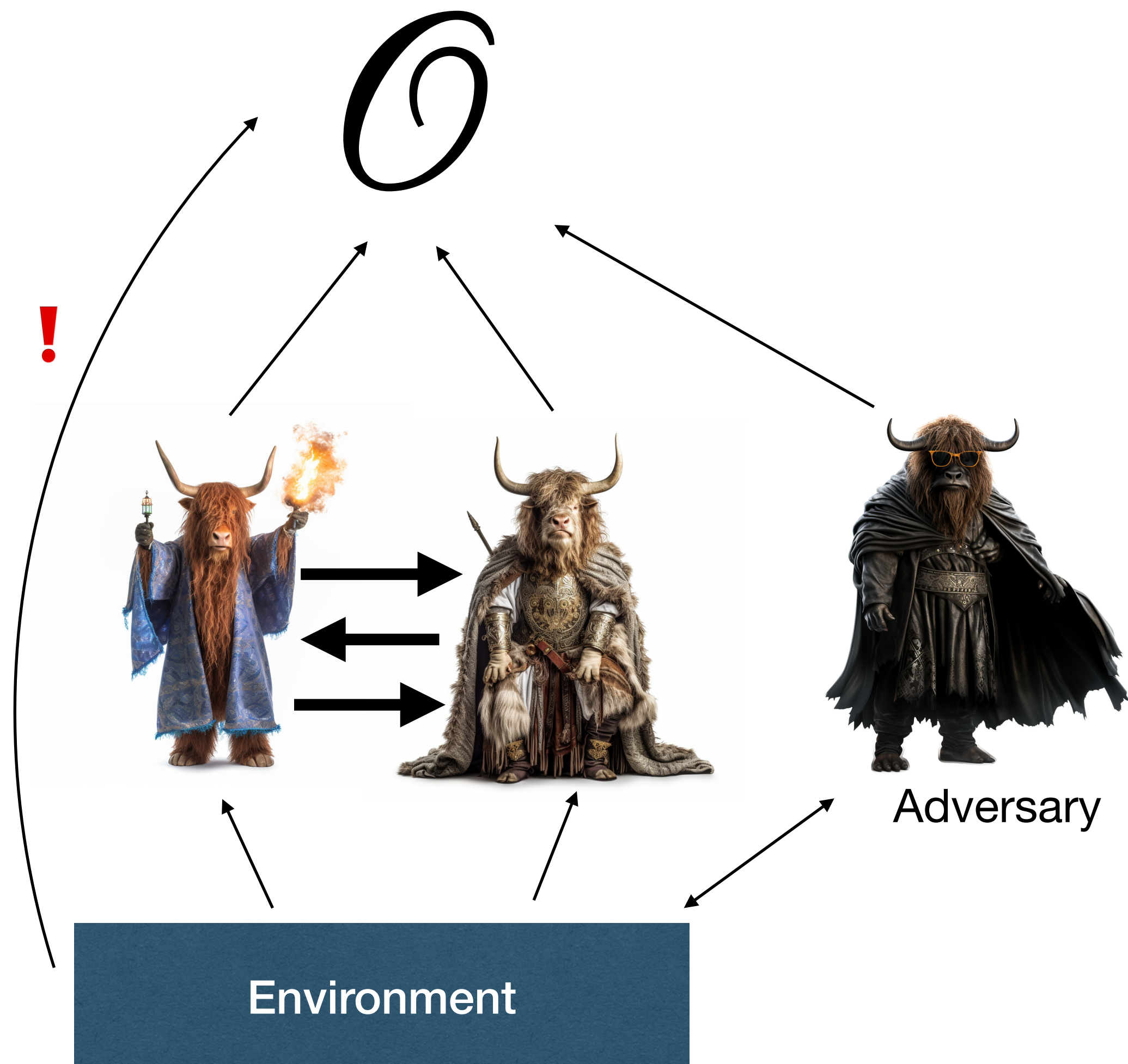
Environment

What we want: Global $\mathcal{O}$

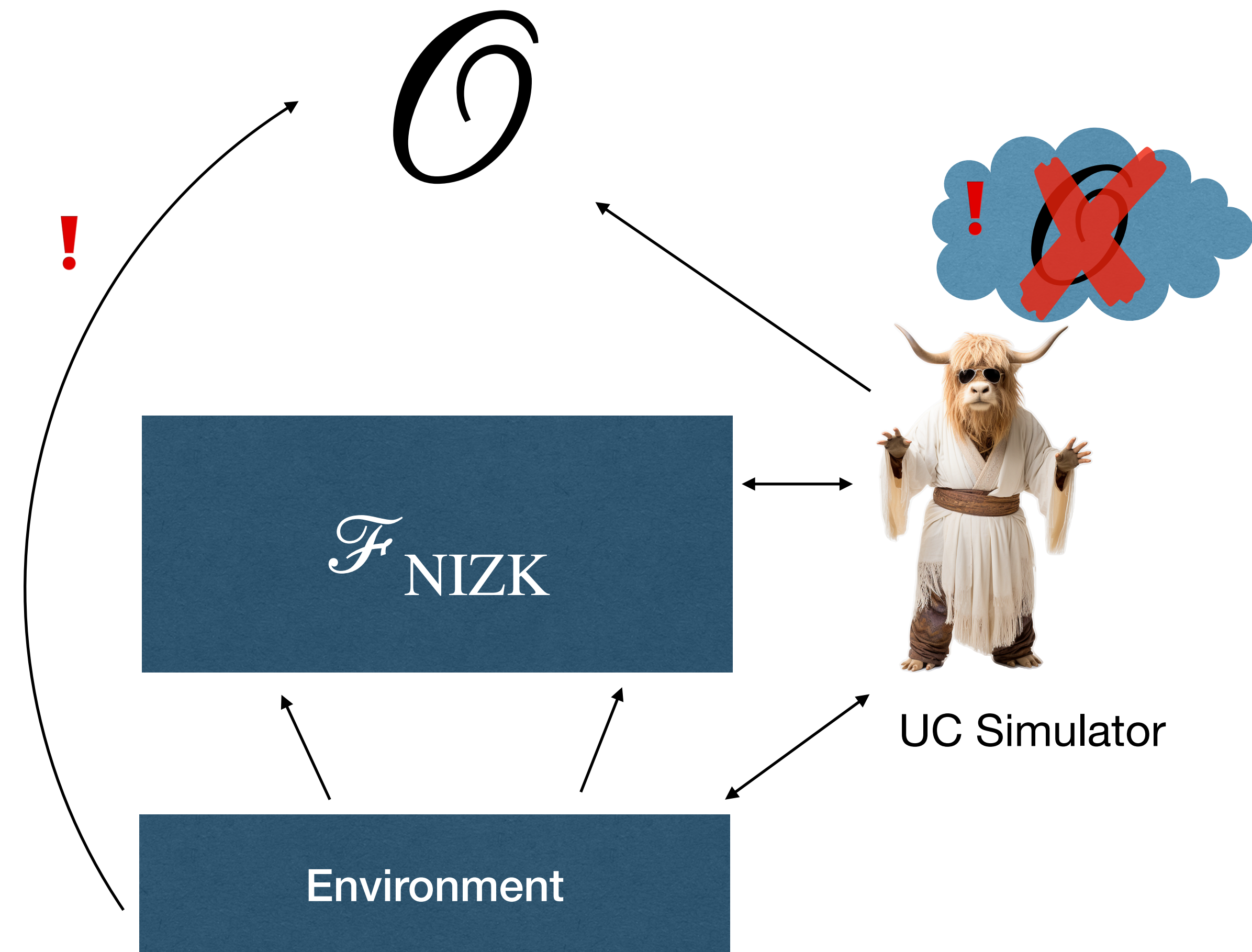


UC modeling with global $\mathcal{O}$

Real world



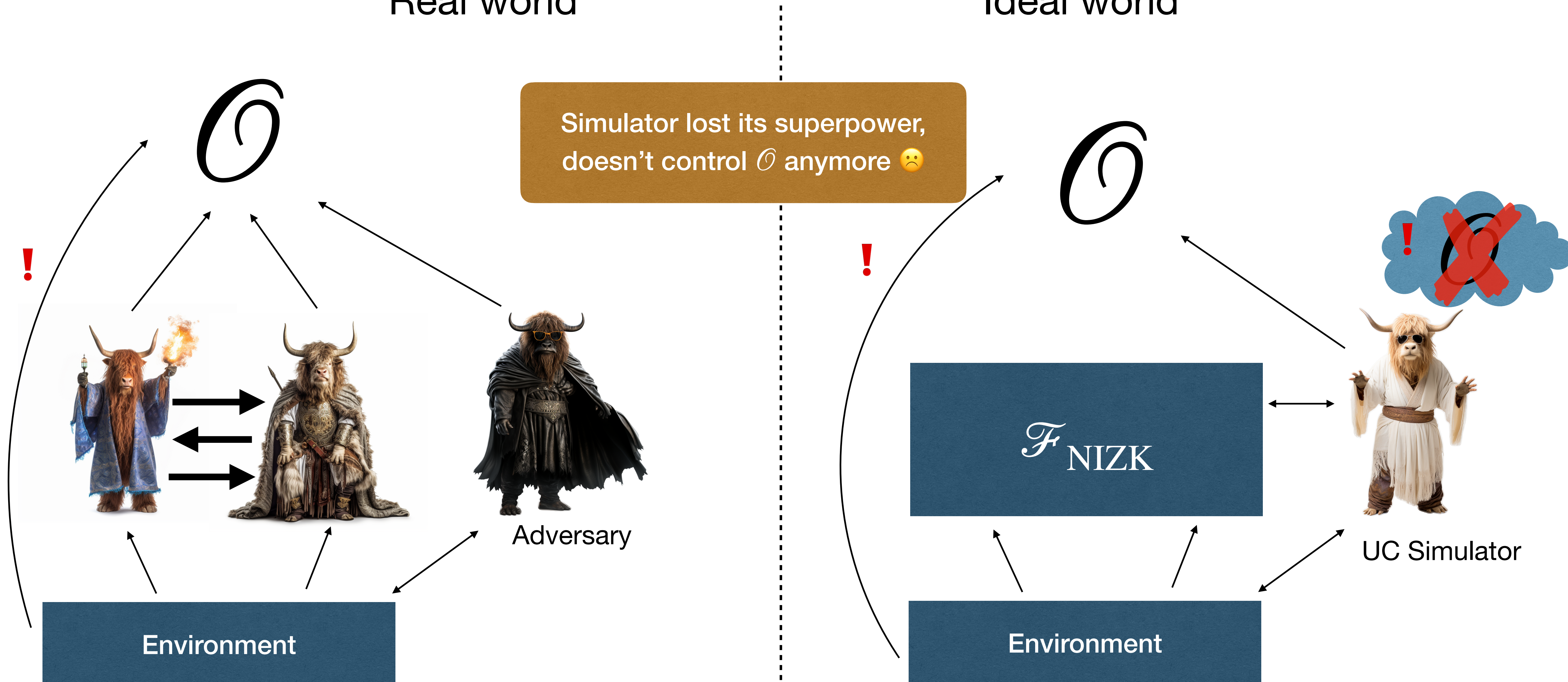
Ideal world



UC modeling with global $\mathcal{O}$

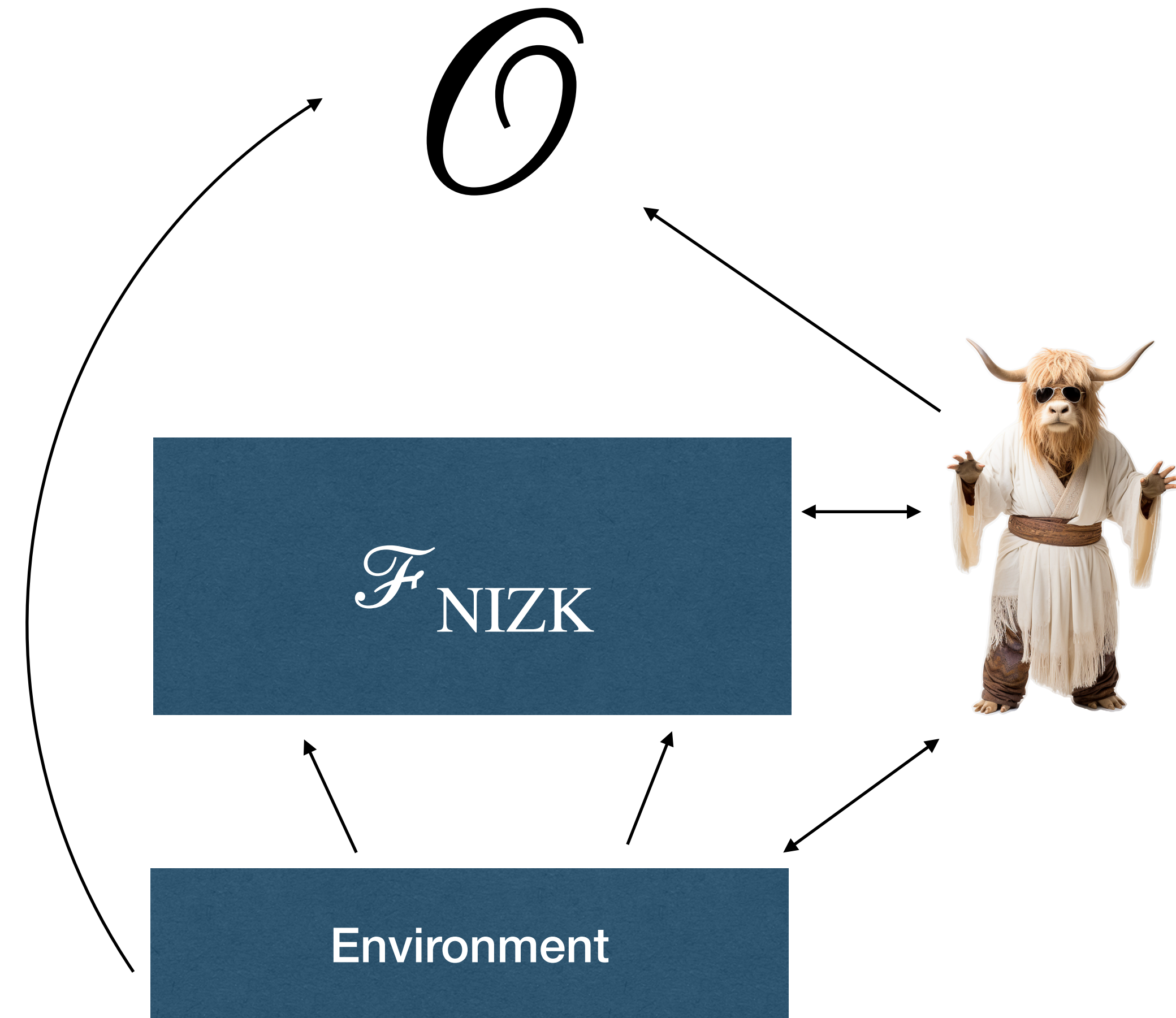
Real world

Ideal world



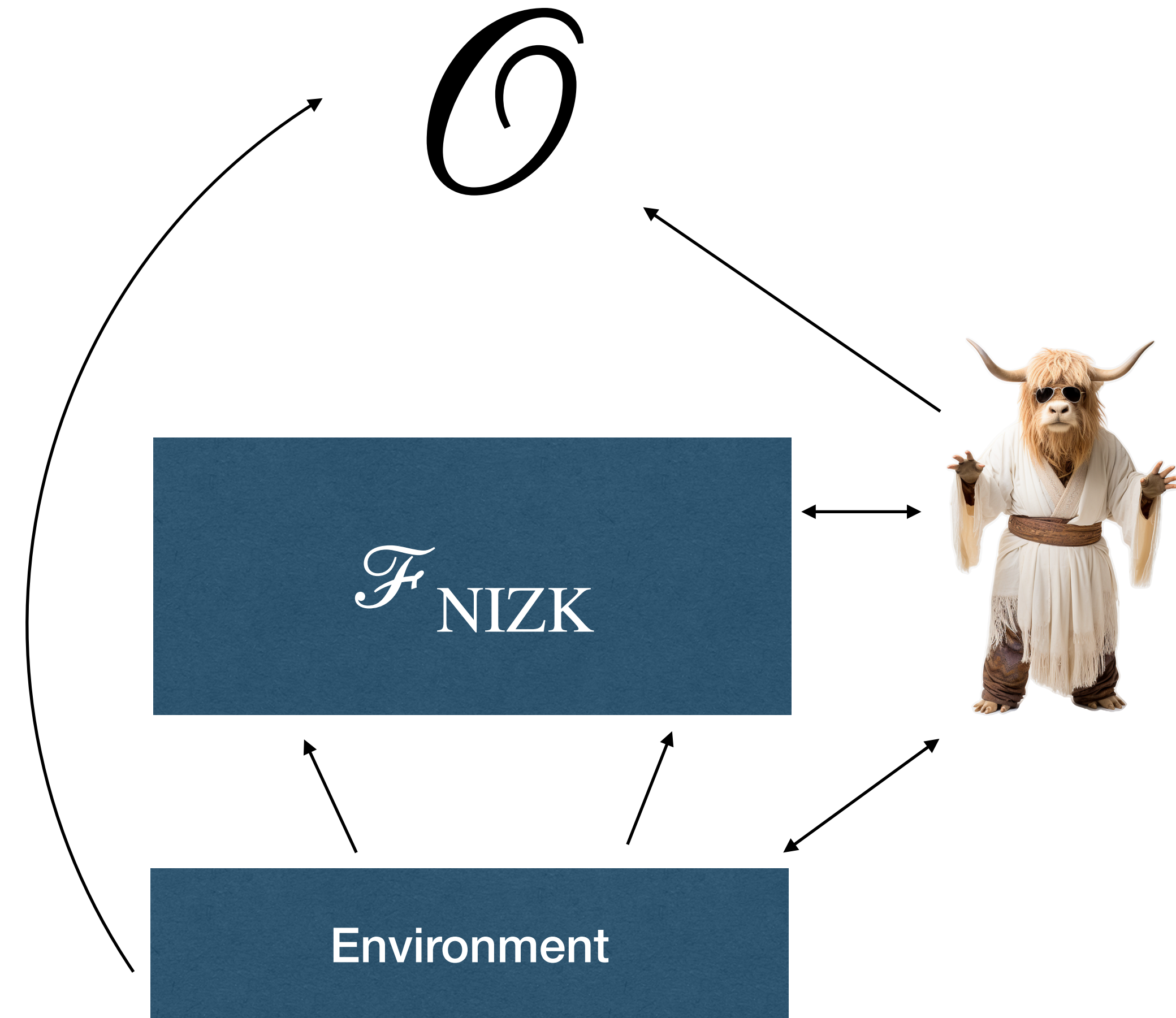
What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing.
Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers.
Impossibilities.



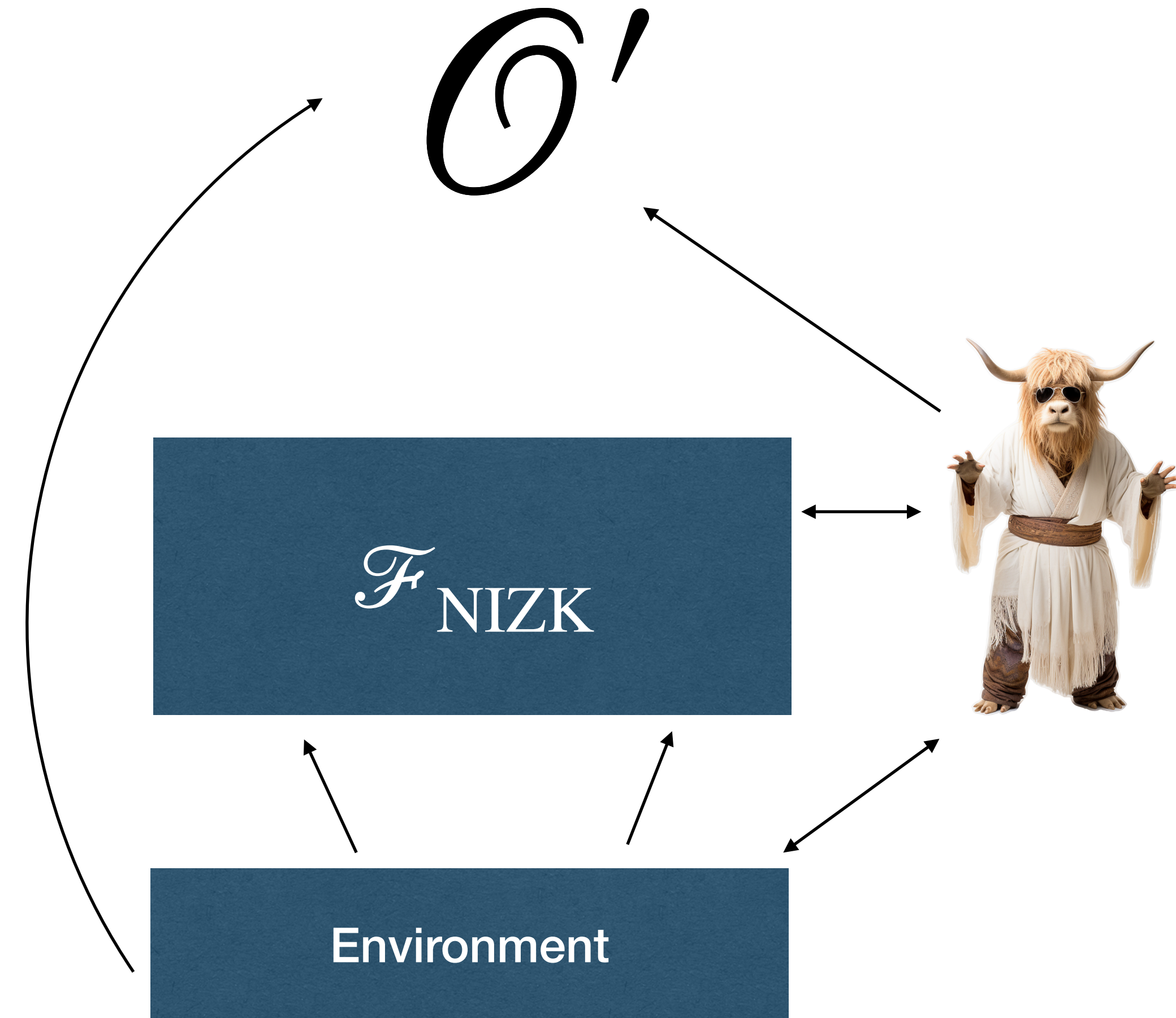
What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing. Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers. Impossibilities.
- 💡 Define some **in-between**?



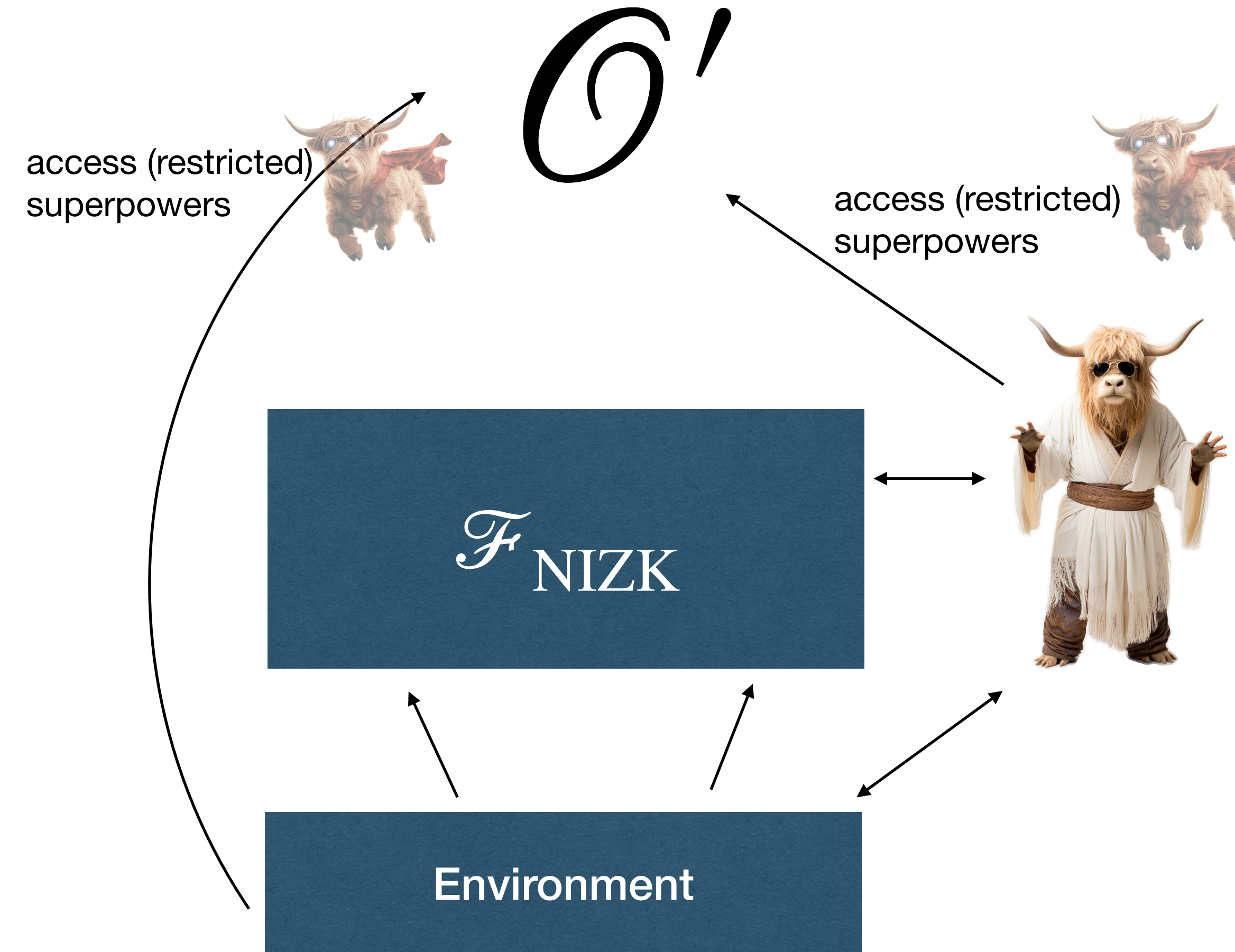
What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing. Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers. Impossibilities.
- 💡 Define some **in-between**?
- **Global** $\mathcal{O}'$ — still shared!



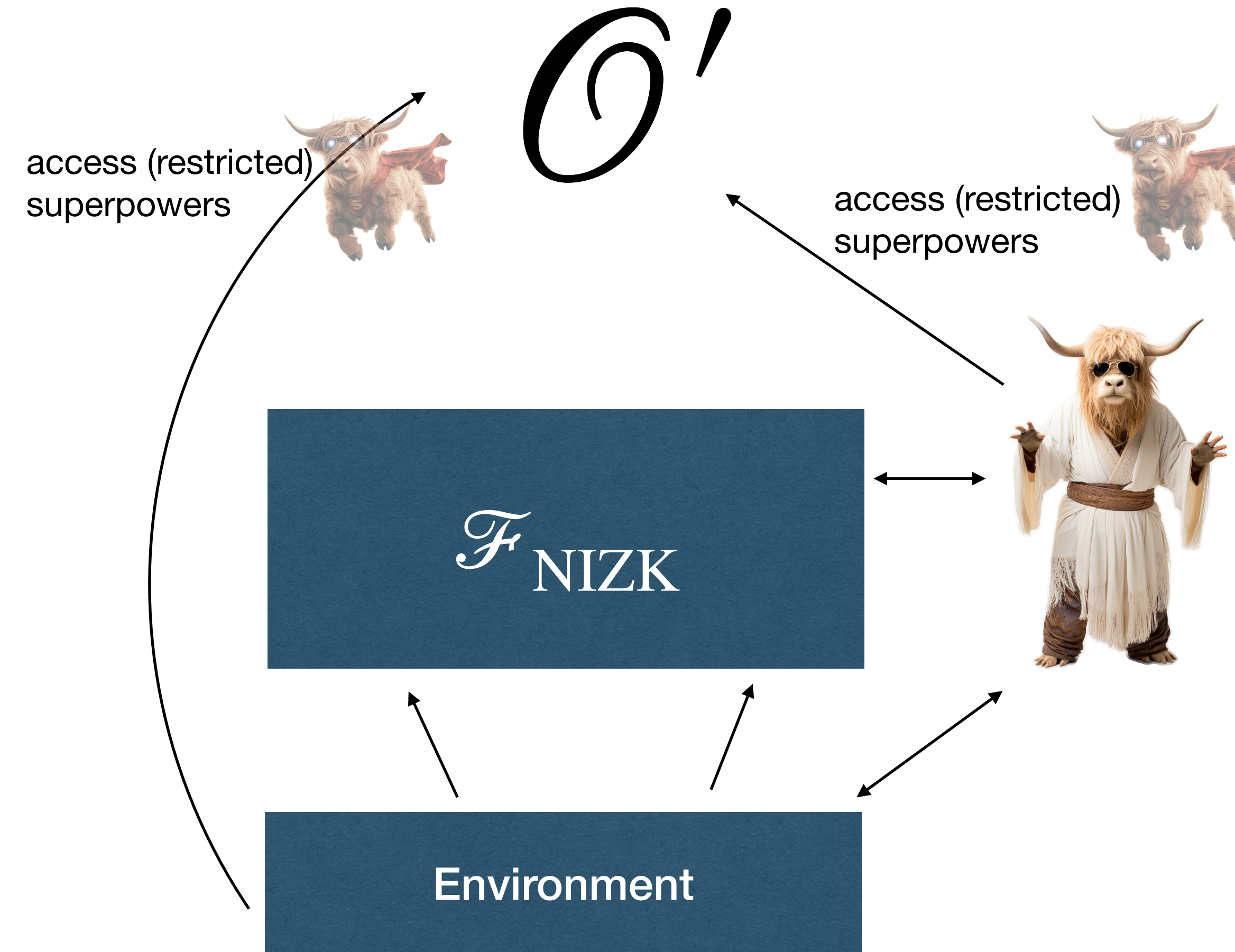
What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing. Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers. Impossibilities.
- 💡 Define some **in-between**?
 - **Global** $\mathcal{O}'$ — still shared!
 - But: **explicitly model superpowers** in $\mathcal{O}'$



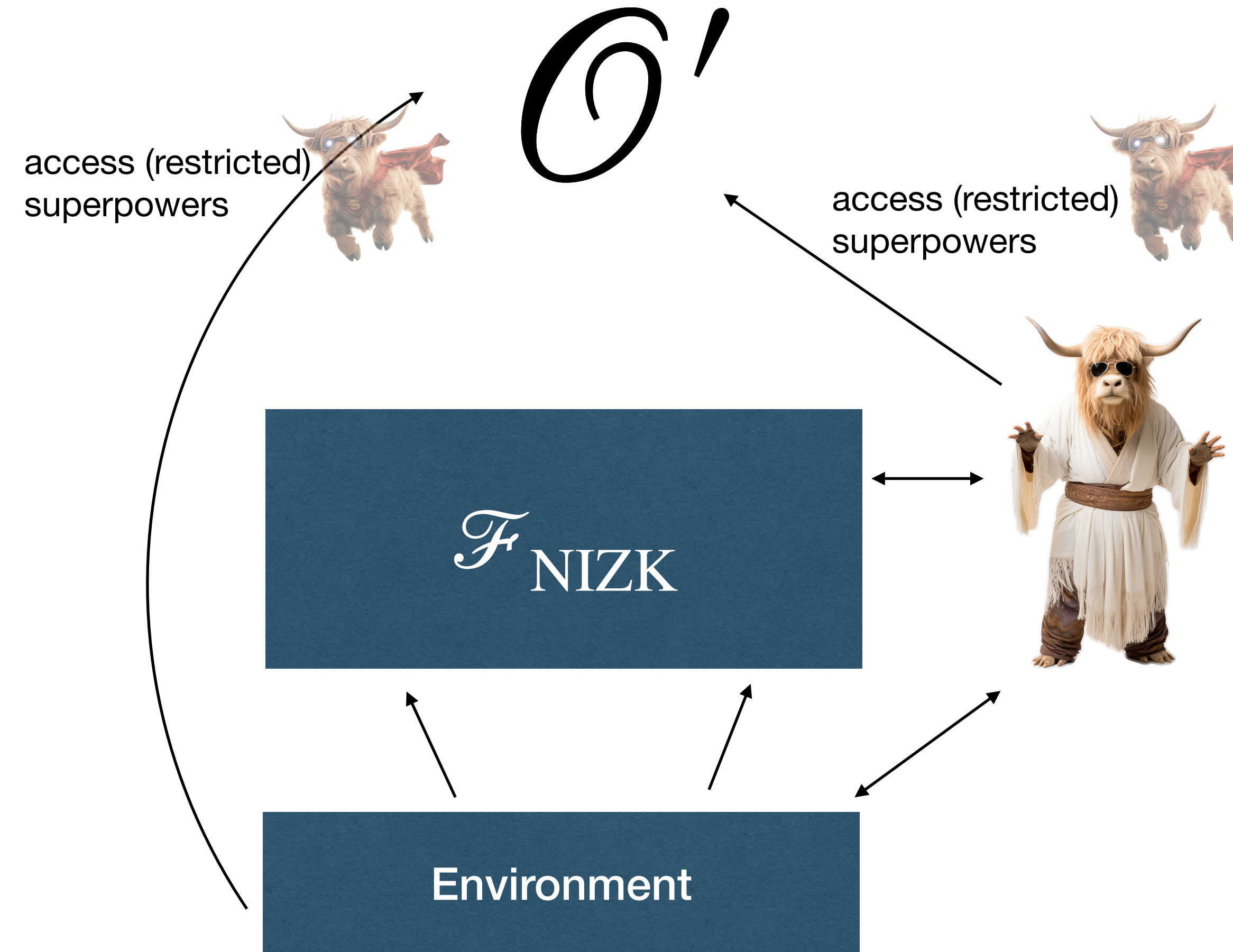
What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing. Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers. Impossibilities.
- 💡 Define some **in-between**?
 - **Global** $\mathcal{O}'$ — still shared!
 - But: **explicitly model superpowers** in $\mathcal{O}'$
 - **Enough** superpower for **UC sim** to simulate/extract



What to do?

- 🙅 **Local** $\mathcal{O}$: does not model sharing. Unrealistic.
- 🙅 **Global** $\mathcal{O}$: UC simulator loses superpowers. Impossibilities.
- 💡 Define some **in-between**?
 - **Global** $\mathcal{O}'$ — still shared!
 - But: **explicitly model superpowers** in $\mathcal{O}'$
 - **Enough** superpower for **UC sim** to simulate/extract
 - **Not so much** superpower that **environment** can simulate/extract.



Superpower 1: Global Random Oracles

From
*Practical UC security
with a global random oracle*

Ran Canetti, Abhishek Jain, Alessandra Scafuro
CCS 2014

*The Wonderful World
of Global Random Oracles*

Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven
Eurocrypt 2018



Random oracle model

```
eval( $x$ ):  
If  $Q(x) = \perp$  :  
     $Q(x) \leftarrow \{0,1\}^n$   
return  $Q(x)$ 
```



Pass's transform

$(a, 0, r_0), (a, 1, r_1)$
 $\leftarrow \Sigma . \text{Prove}(x, w)$



$$\begin{aligned} \mathcal{C}_0 &= \text{RO}(x, (a, 0, r_0), \text{nonce}_0) \\ \mathcal{C}_1 &= \text{RO}(x, (a, 1, r_1), \text{nonce}_1) \end{aligned}$$

$$c \leftarrow \{0, 1\}$$

$$r_c, \text{nonce}_c$$



Extractor



Check $\Sigma . \text{Verify}(x, a, c, r_c)$

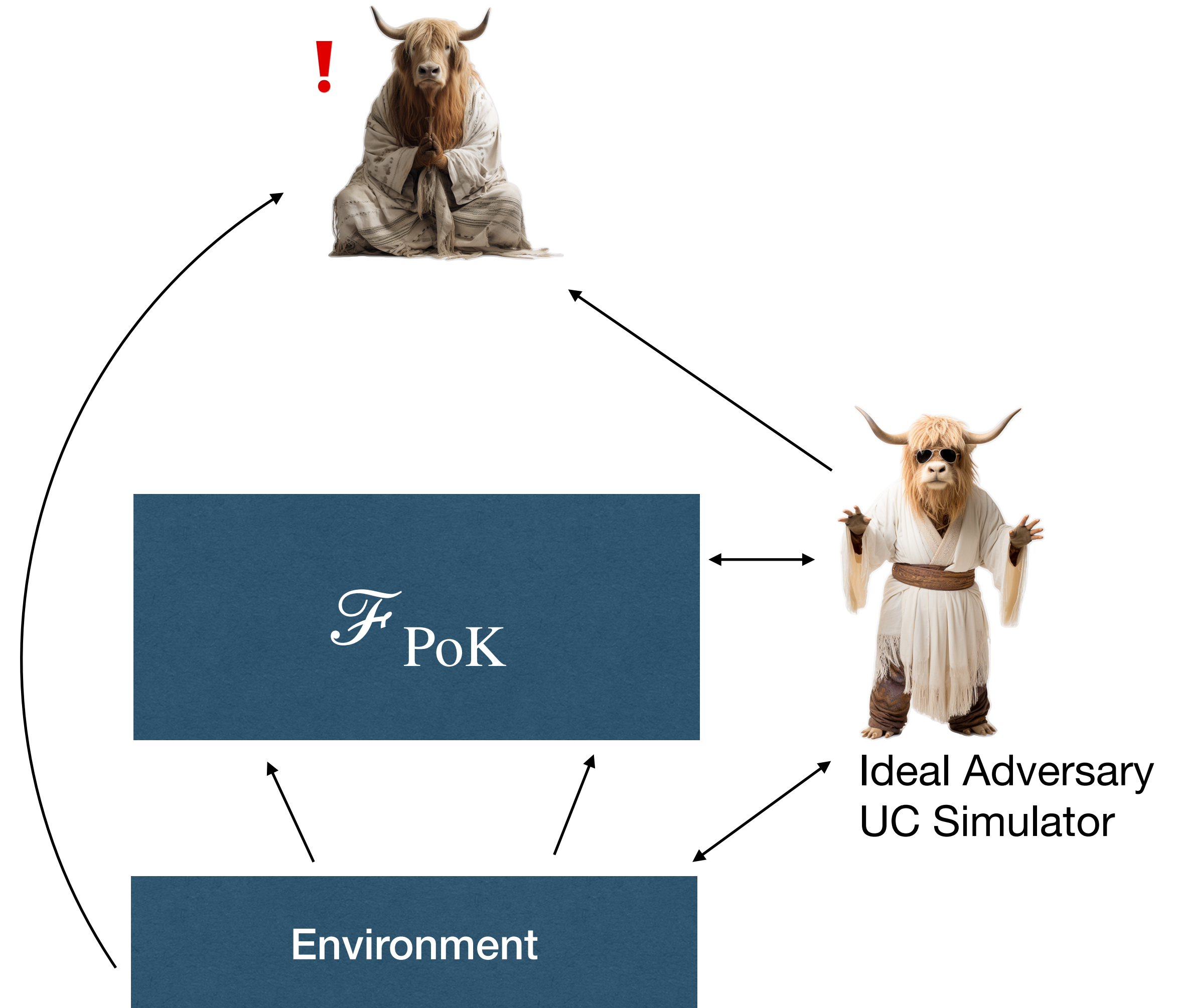
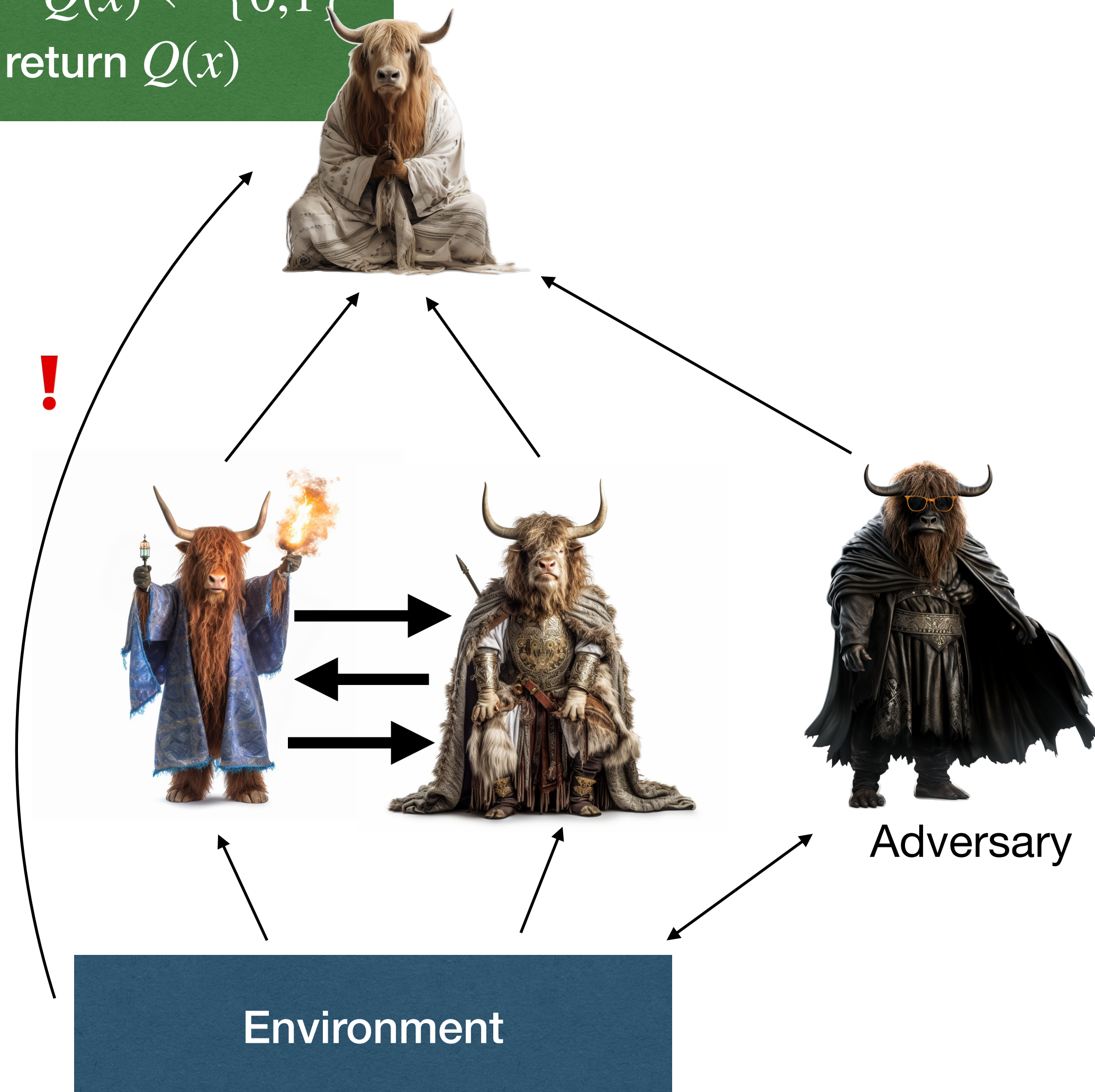
Check $\mathcal{C}_c = \text{RO}(x, (a, c, r_c), \text{nonce}_c)$

Two worlds with global ROM

Real world

Ideal world

```
eval(x):  
If  $Q(x) = \perp$  :  
   $Q(x) \leftarrow \{0,1\}^n$   
return  $Q(x)$ 
```



Two worlds with global ROM

Real world

Ideal world

Simulator lost superpower of
observing all RO queries 😞

Environment

Adversary

Environment

Ideal Adversary
UC Simulator

The *observable* global ROM



The *observable* global ROM

💡 **Idea:** Add observation capabilities to the ROM functionality.



Fully observable global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

Add x to Ω

return $Q(x)$

getObs():

return Ω

💡 **Idea:** Add observation capabilities to the ROM functionality.



Fully observable global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

Add x to Ω

return $Q(x)$

getObs():

return Ω

💡 **Idea:** Add observation capabilities to the ROM functionality.

Okay. Too much! No chance for ZK.
Everyone sees everything always.



Fully observable global ROM

```
eval(x):  
  If  $Q(x) = \perp$  :  
     $Q(x) \leftarrow \{0,1\}^n$   
  Add  $x$  to  $\Omega$   
  
  return  $Q(x)$   
  
getObs():  
  return  $\Omega$ 
```

Okay. Too much! No chance for ZK.
Everyone sees everything always.



💡 **Idea:** Add observation capabilities to the ROM functionality.

📋 Requirements

- *For PoK:* Simulator **must be** able to observe environment's queries.
- *For ZK:* Environment **must not be** able to observe honest parties' queries.



Dream observable global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

if caller is environment:

Add x to Ω

return $Q(x)$

getObs():

return Ω



Idea: Add observation capabilities to the ROM functionality.



Requirements

- *For PoK:* Simulator **must be** able to observe environment's queries.
- *For ZK:* Environment **must not be** able to observe honest parties' queries.





Dream observable global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

if caller is environment:

Add x to Ω

return $Q(x)$

getObs():

return Ω



Idea: Add observation capabilities to the ROM functionality.



Requirements

- *For PoK:* Simulator **must be** able to observe environment's queries.
- *For ZK:* Environment **must not be** able to observe honest parties' queries.

Cannot check this. *Doesn't make sense* from G-ROM PoV





Dream observable global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

if caller is environment:

Add x to Ω

return $Q(x)$

getObs():

return Ω

Cannot check this. *Doesn't make sense* from G-ROM PoV



Idea: Add observation capabilities to the ROM functionality.



Requirements

- *For PoK*: Simulator **must be** able to observe environment's queries.
- *For ZK*: Environment **must not be** able to observe honest parties' queries.



Solution: Domain separation

The *observable* global ROM

eval(x):

If $Q(x) = \perp$:

$Q(x) \leftarrow \{0,1\}^n$

Parse $x = (s, x')$

if caller's session is not s :

Add x to Ω

return $Q(x)$

getObs():

return Ω

💡 **Idea:** Add observation capabilities to the ROM functionality.

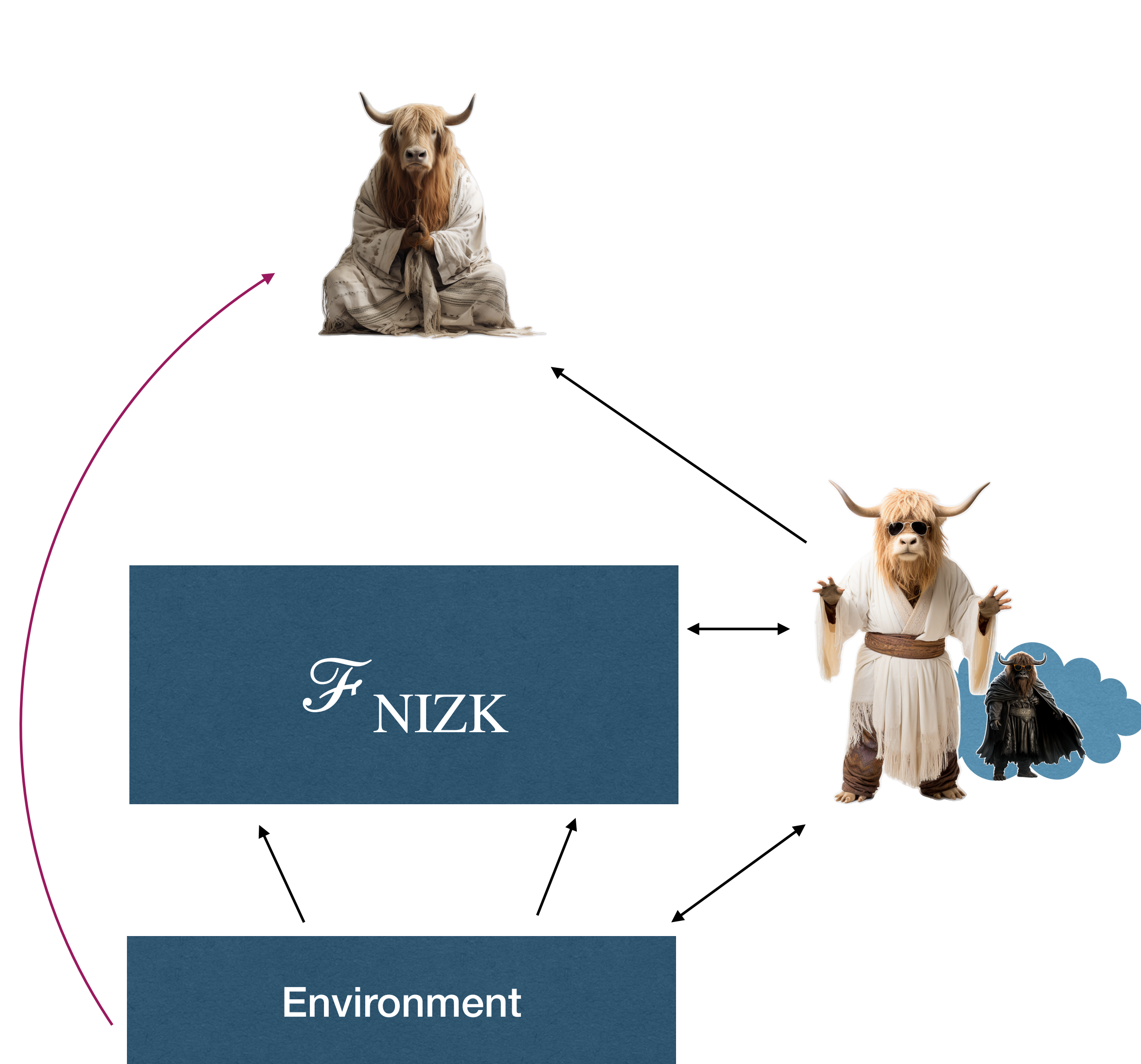
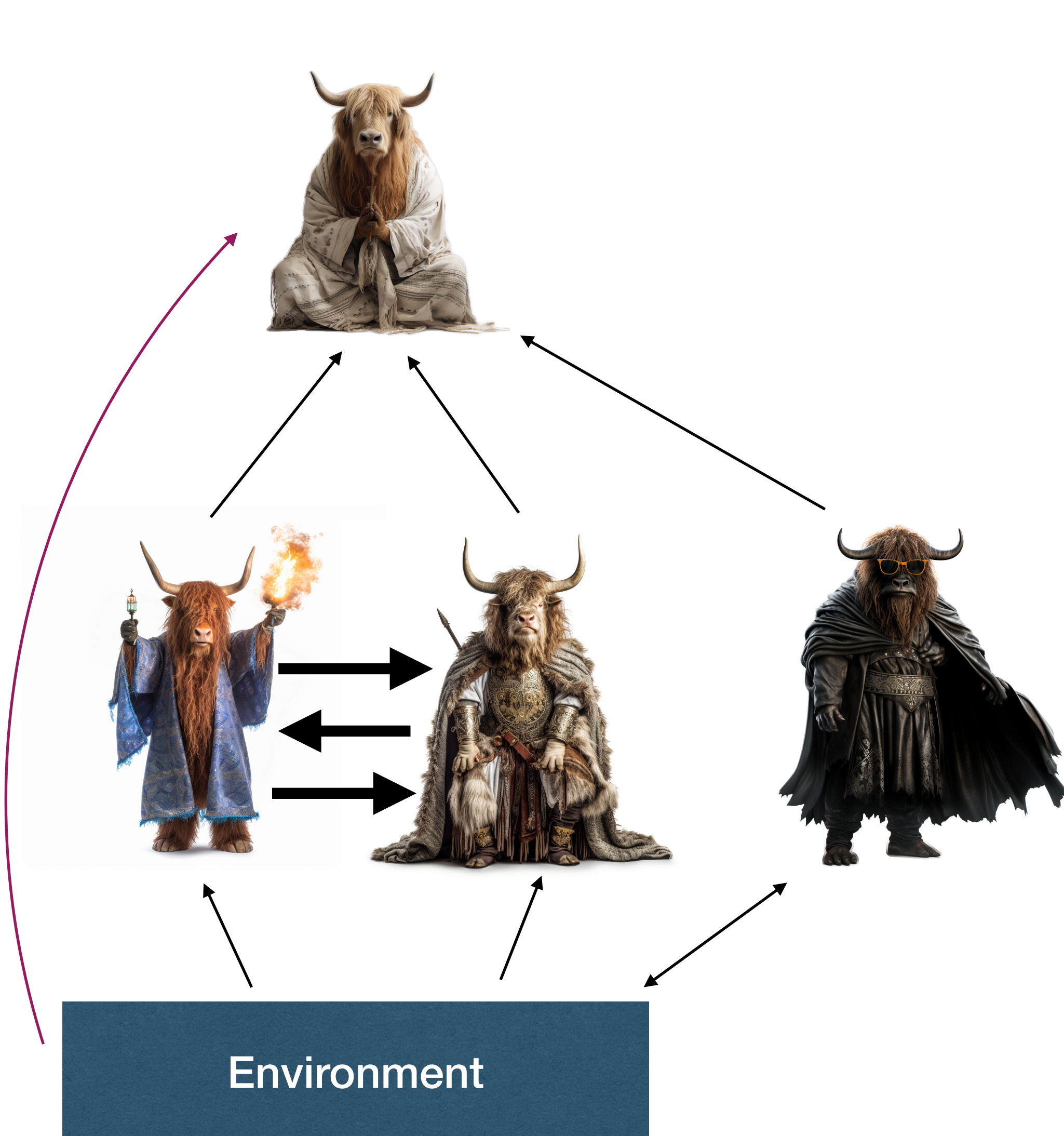
📋 Requirements

- *For PoK:* Simulator **must be** able to observe environment's queries.
- *For ZK:* Environment **must not be** able to observe honest parties' queries.

🦆 **Solution:** Domain separation

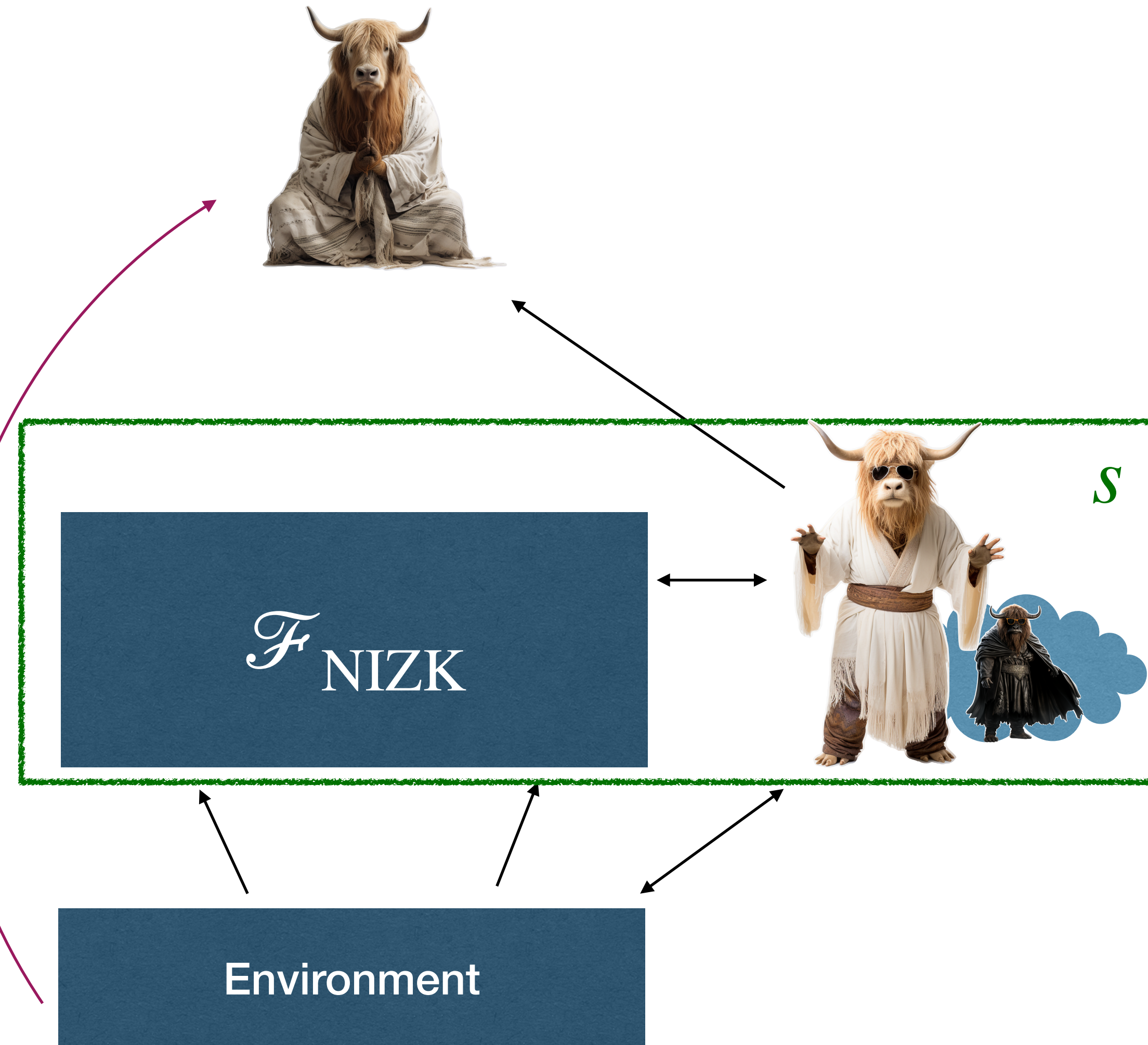
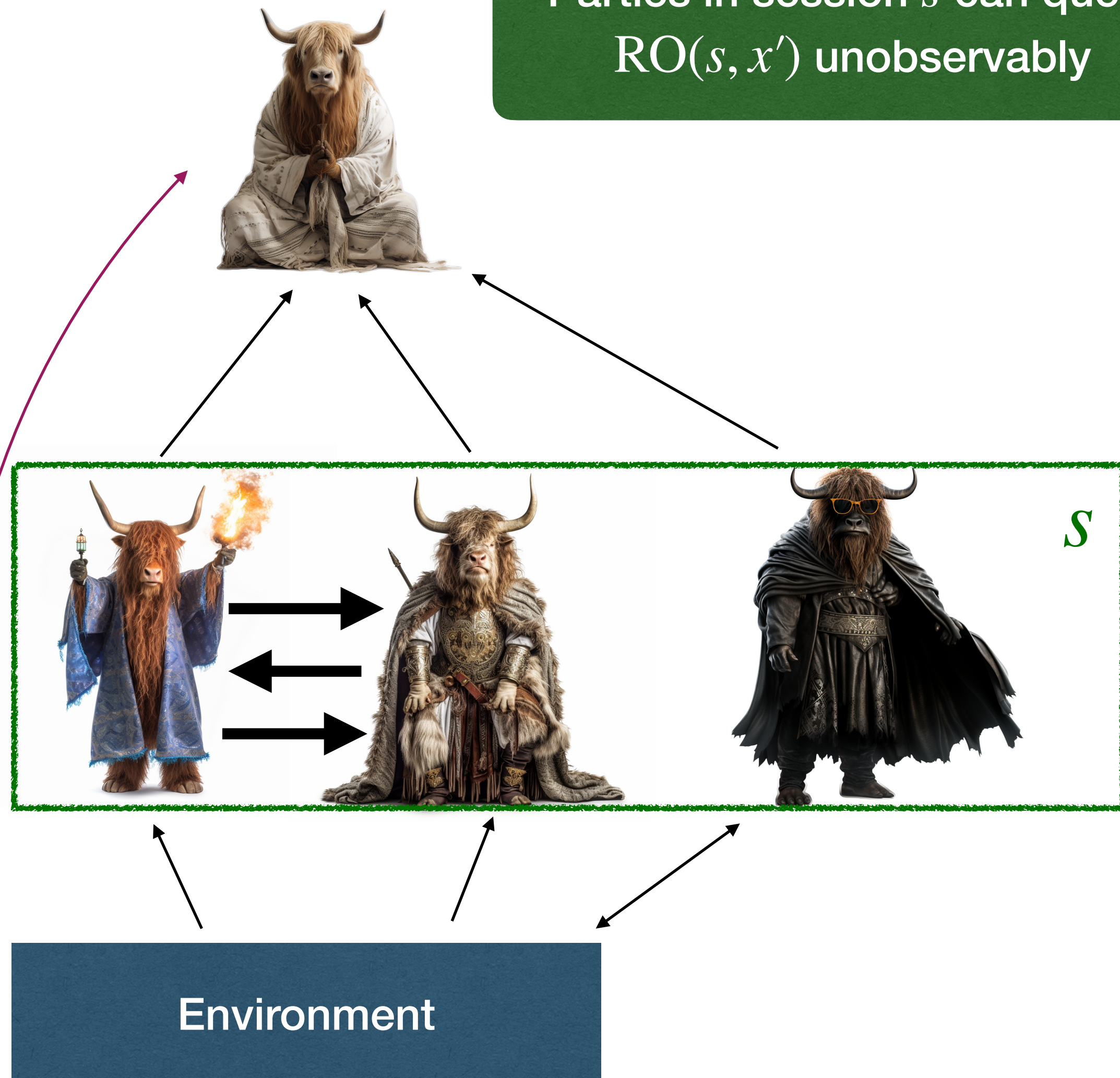


Two worlds with global *observable* ROM

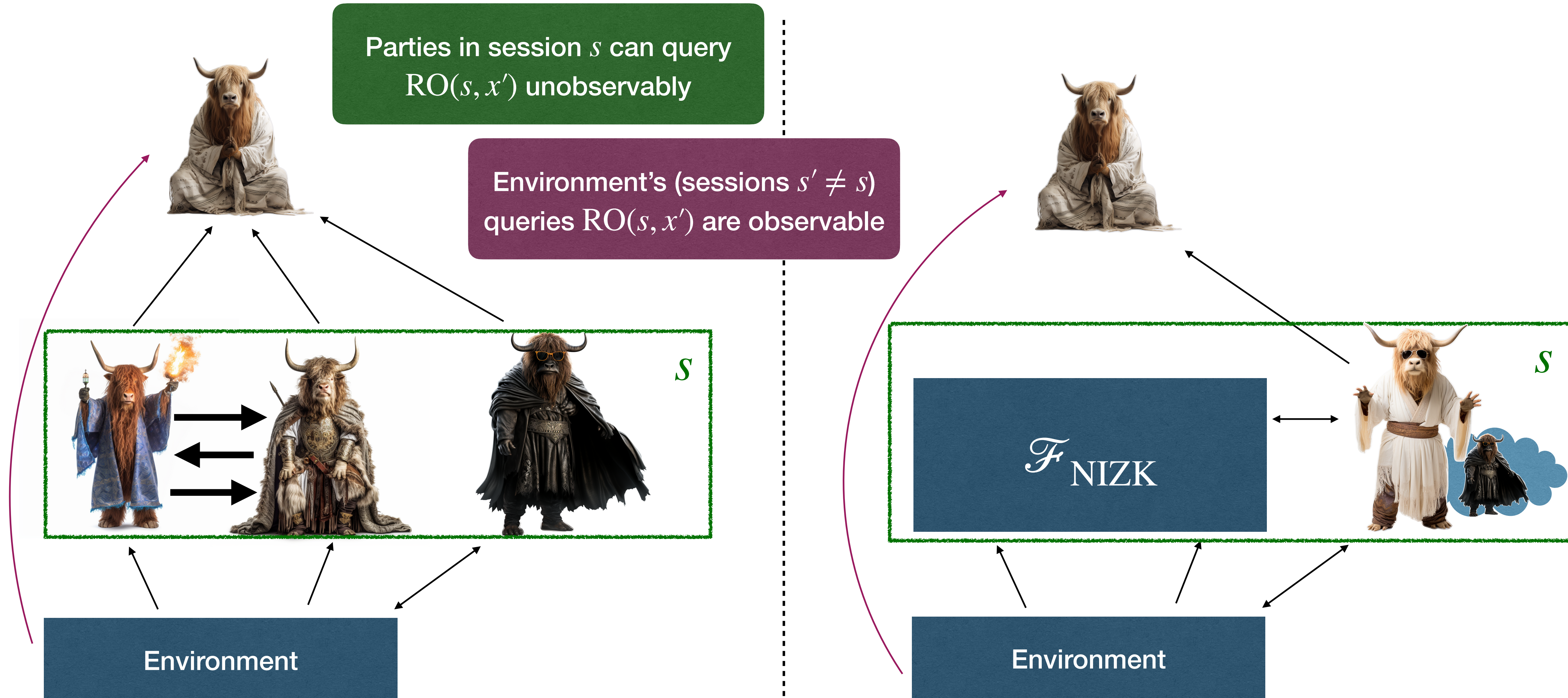


Two worlds with global *observable* ROM

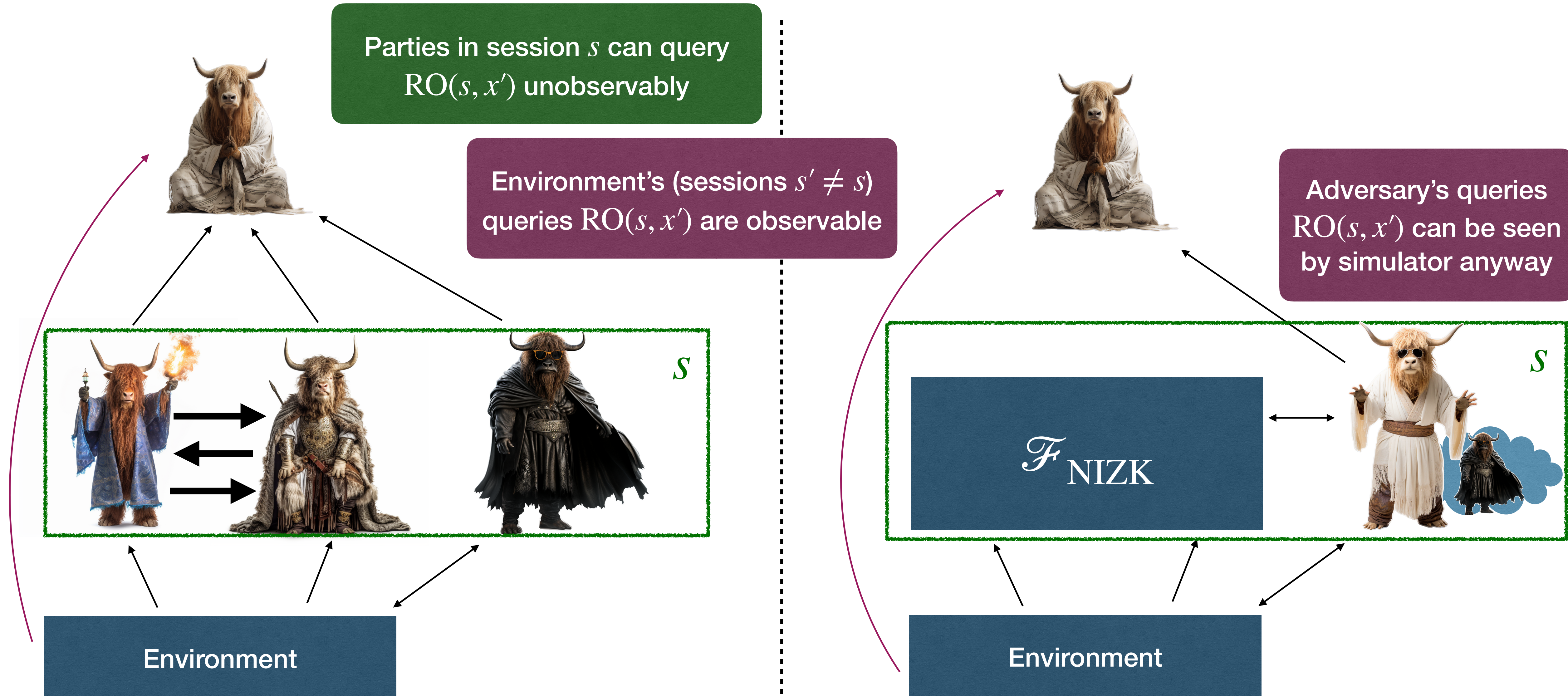
Parties in session s can query $\text{RO}(s, x')$ unobservably



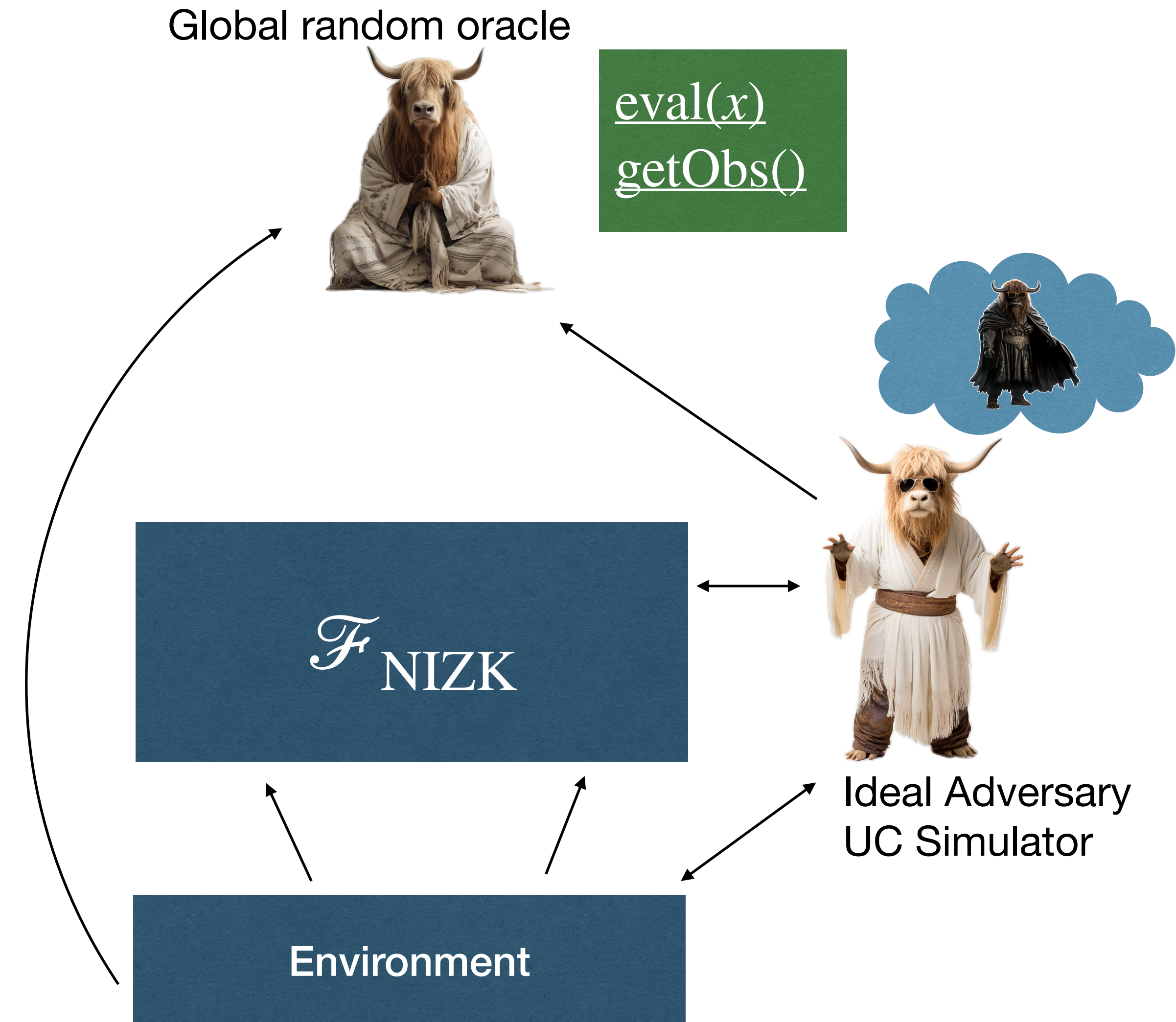
Two worlds with global *observable* ROM



Two worlds with global *observable* ROM

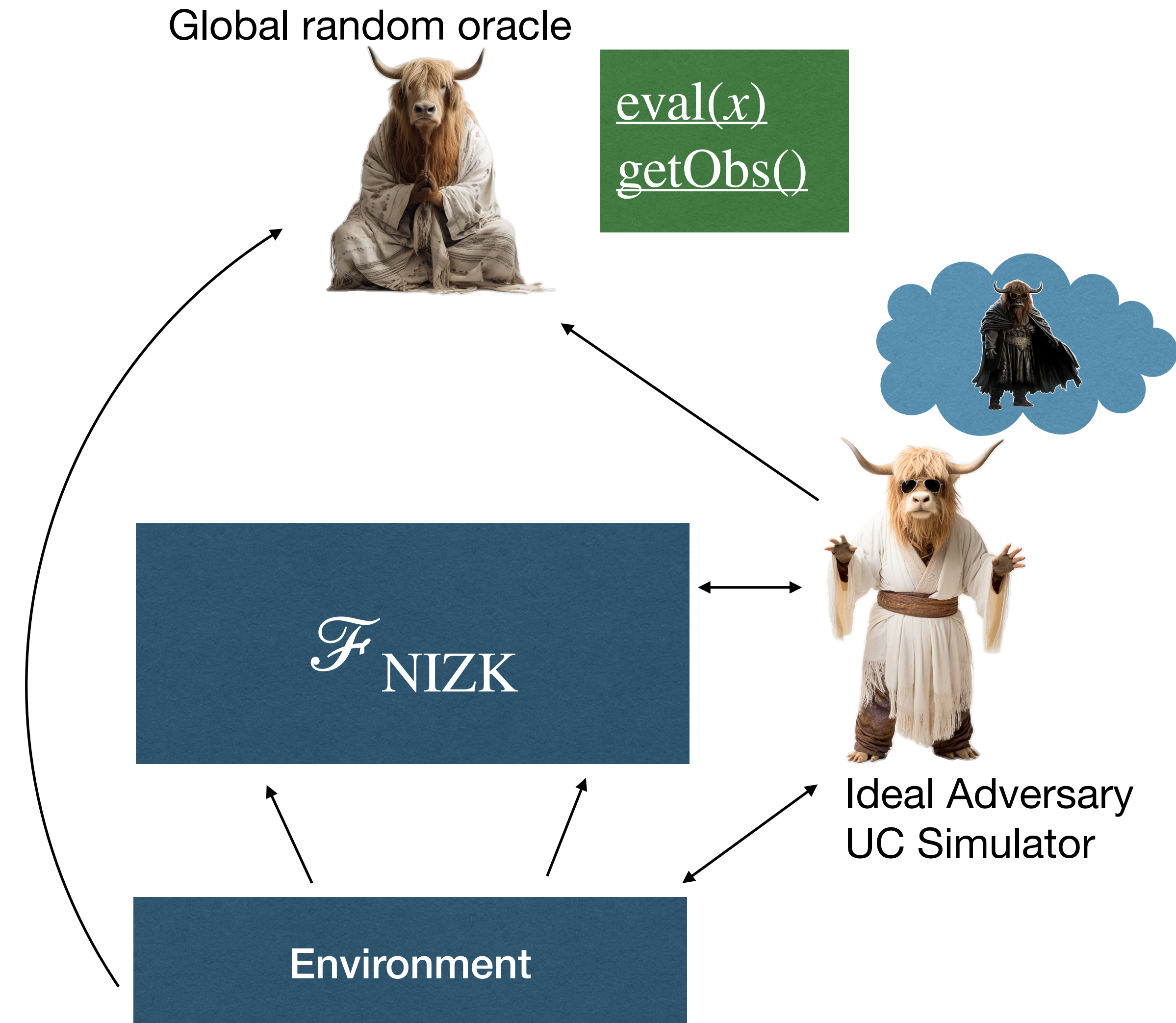


Summary part 1



Summary part 1

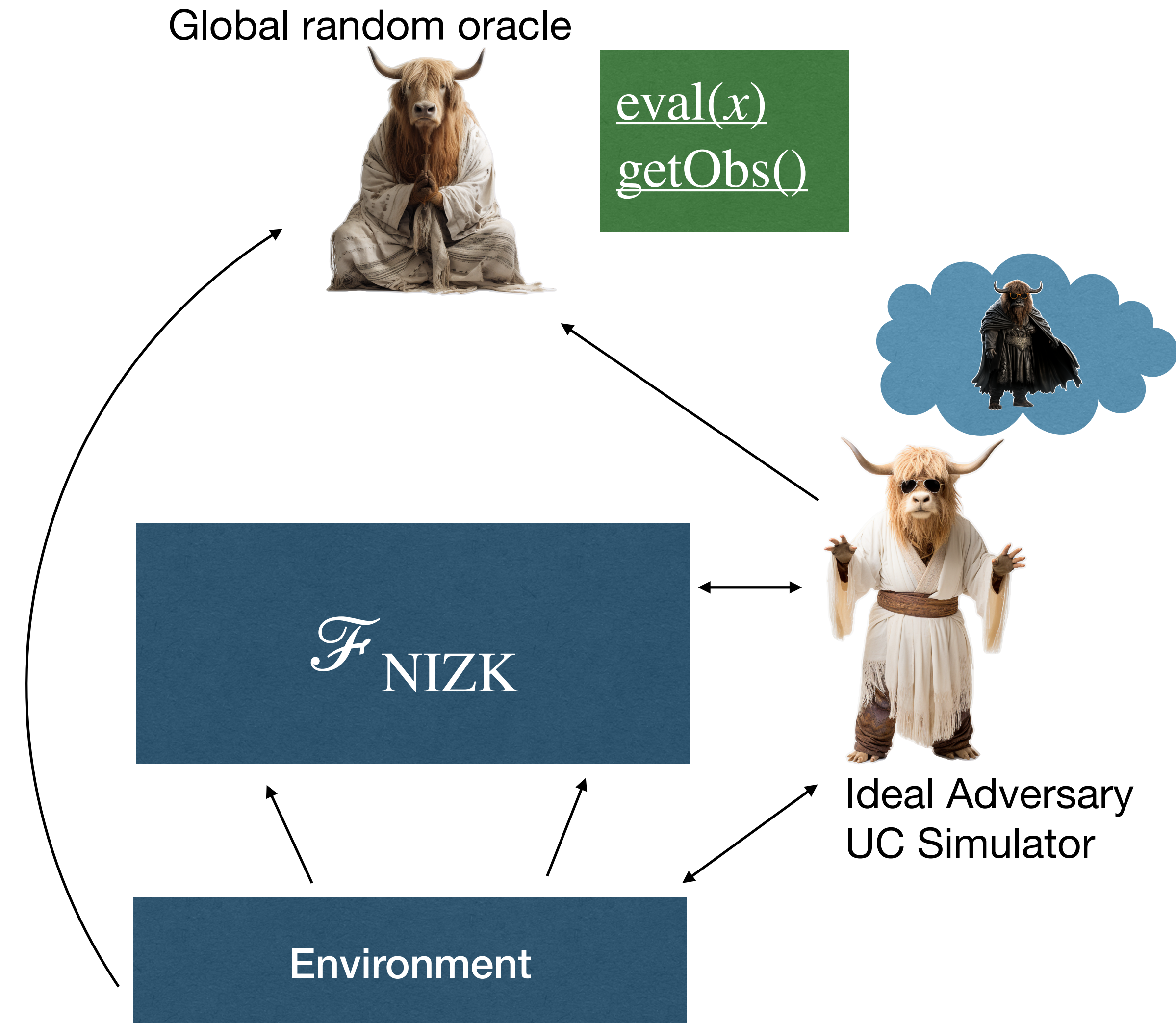
-  **Global ROM** models composition with shared RO



Summary part 1

- 🌐 **Global ROM** models composition with shared RO
- 👁️ Need to explicitly **model observability**
- 😇 **Good:** Session s calls $\text{RO}(s, x')$
- 😈 **Bad:** Session $s' \neq s$ calls $\text{RO}(s, x')$

⇒ Observable



Summary part 1

- 🌐 **Global ROM** models composition with shared RO

- 👁️ Need to explicitly **model observability**

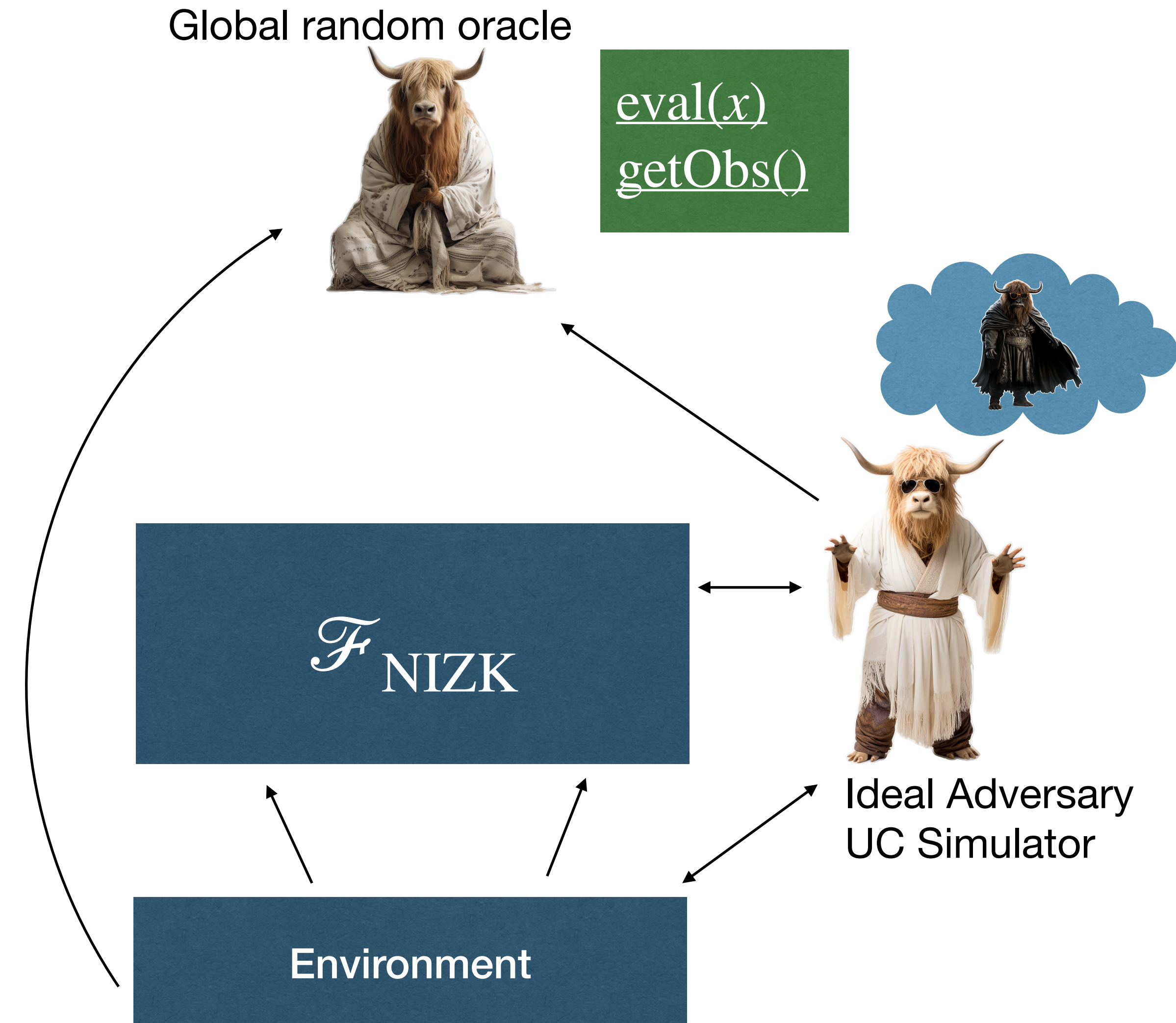
- 😇 **Good:** Session s calls $RO(s, x')$

- 😈 **Bad:** Session $s' \neq s$ calls $RO(s, x')$

⇒ Observable

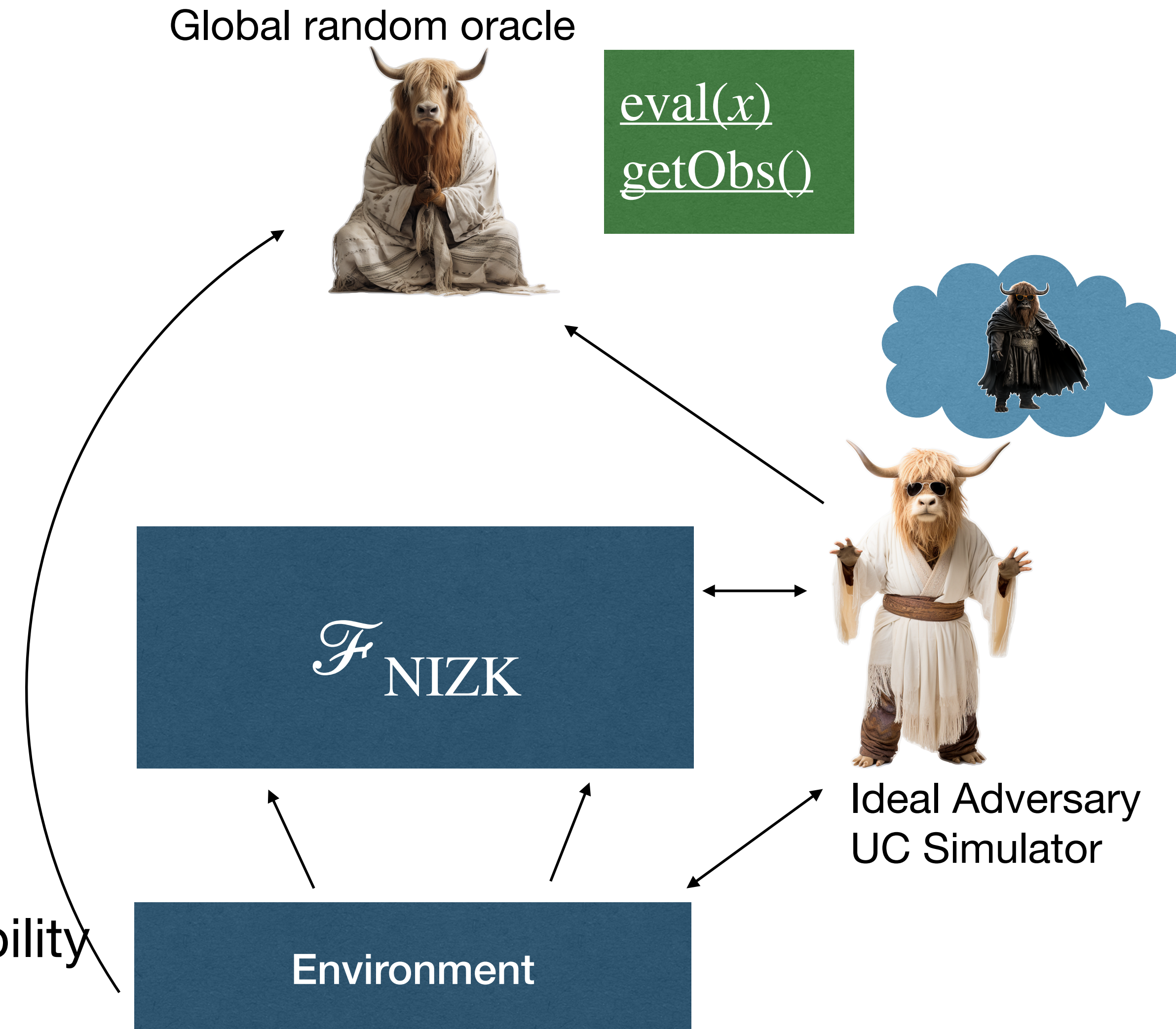
- 🎉 Get NIZKPoK in UC via Fischlin/Pass (+crs for simulation)

Lysyanskaya, Rosenbloom; ia.cr/2022/290

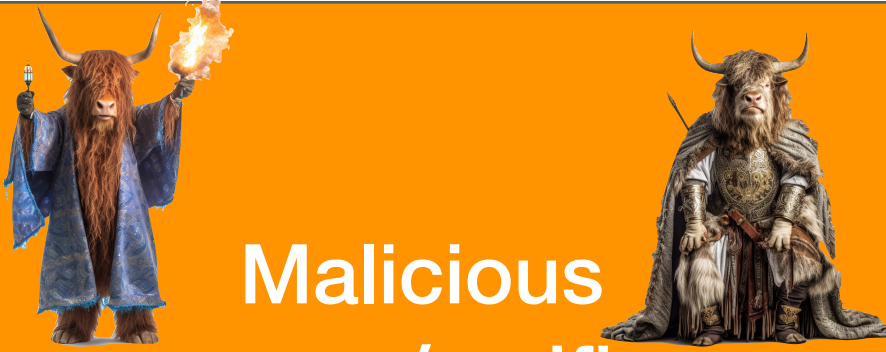


Summary part 1

- 🏠 **Global ROM** models composition with shared RO
- 👁️ Need to explicitly **model observability**
- 😇 **Good:** Session s calls $\text{RO}(s, x')$
- 😈 **Bad:** Session $s' \neq s$ calls $\text{RO}(s, x')$
 $\Rightarrow$ Observable
- 🎉 Get NIZKPoK in UC via Fischlin/Pass (+crs for simulation)
Lysyanskaya, Rosenbloom; ia.cr/2022/290
- 🧑 **Learned:** need domain-separation for composability
("implicit" in one-shot games/hybrid UC)



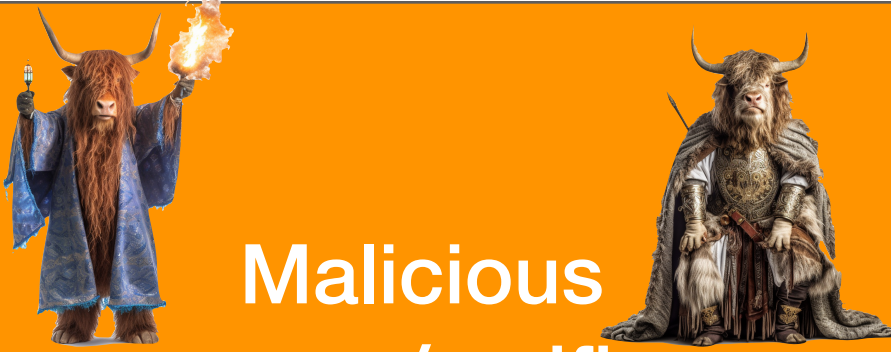


 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations

 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations



Impossible in UC
(doesn't generally compose)

 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations



Impossible in UC
(doesn't generally compose)

... it's complicated ...

 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations



Impossible in UC
(doesn't generally compose)

... it's complicated ...

Solved previously
Wonderful world paper
ia.cr/2018/165

 Strategy	 Real life	 Superpower of Simulator/Extractor	 Malicious prover/verifier
 Rewinding	Prover/verifier runs protocol from start to finish	Re-run party multiple times with different inputs	Cannot re-run party
 Common Reference String	Trusted party generates and publishes CRS	Generate CRS, embed trapdoor	Does not control CRS, doesn't know trapdoor
 Random oracle model	Use strong hash function (SHA-2/3, Poseidon, ...)	Can see hash queries and decide hash values	Doesn't see hash queries, cannot predict hash values
 Generic group model / AGM / knowledge assumption	Use strong group (elliptic curve groups)	Can see group operations	Cannot see group operations



Impossible in UC
(doesn't generally compose)

... it's complicated ...

Solved previously
Wonderful world paper
ia.cr/2018/165

Now also solved!
Brave new world paper
ia.cr/2024/818

Superpower 2: Generic groups

The resource behind Groth16 extraction



Now also solved!
Brave new world paper
ia.cr/2024/818

The generic group model



The generic group model

- 🎯 Goal: model “**idealized**” group with no “extra” structure (just group operations[, pairing]).
- Similar to random oracles, which model “idealized” hash function with no structure.



The generic group model

- 🎯 Goal: model “**idealized**” group with no “extra” structure (just group operations[, pairing]).
 - Similar to random oracles, which model “idealized” hash function with no structure.
- 💡 Idea: group elements get **random encoding** (= no structure), but **oracle $\mathcal{O}$** enables group ops.



The generic group model

- 🎯 Goal: model “**idealized**” group with no “extra” structure (just group operations[, pairing]).
 - Similar to random oracles, which model “idealized” hash function with no structure.
- 💡 Idea: group elements get *random encoding* (= no structure), but **oracle $\mathcal{O}$ enables group ops.**
- 🐉 Superpower: **observe** all group ops.



The generic group model

$\mathcal{G} =$

```
private random injective  $\tau : \mathbb{G} \rightarrow S$   
public generator  $g$ 
```

```
op( $g_1, g_2$ ):  
return  $\tau(\tau^{-1}(g_1) + \tau^{-1}(g_2))$ 
```



Naive $\mathcal{O}' = \text{observable GGM}$

$\mathcal{O}' =$

```
private random injective  $\tau : \mathbb{G} \rightarrow \mathcal{S}$   
public generator  $g$ 
```

```
op( $g_1, g_2$ ):  
  add ( $g_1, g_2$ ) to public observation list  
  return  $\tau(\tau^{-1}(g_1) + \tau^{-1}(g_2))$ 
```

```
observe():  
  return public observation list
```





CRS: group elements $u_i, \alpha, \beta, \delta$

Witness: wire values $a_i \in \mathbb{Z}_p$



Groth16

Extractor



$$A = \sum_{i=0}^m a_i u_i + \alpha + r\delta$$

$$B = \sum_{i=0}^m a_i v_i + \beta + r'\delta$$

$$C = \dots$$



observe()



[Check some pairing equations on A,B,C]

Design challenges

- 🧑‍🔧 Requirements:



Design challenges

- 🧑‍🔧 Requirements:
- 👁️ Simulator **must see** group operations made by environment
- Required to extract



Design challenges

- 🧑 Requirements:
 - 👁 Simulator **must see** group operations made by environment
 - Required to extract
 - 🚫 Environment **must not see** what group operations simulator makes
 - Would immediately reveal that we simulate



Design challenges

- 🧑 Requirements:
- 👁 Simulator **must see** group operations made by environment
 - Required to extract
- 🙅 Environment **must not see** what group operations simulator makes
 - Would immediately reveal that we simulate
- ⚠ First glance: **Impossible**



Design challenges

- 🧑‍🔧 Requirements:
 - 👁 Simulator **must see** group operations made by environment
 - Required to extract
 - 🚫👁 Environment **must not see** what group operations simulator makes
 - Would immediately reveal that we simulate
- ⚠ First glance: **Impossible**
- 💡 Do **partial observability** via **domain separation**



Design challenges



- 🧑 Requirements:
 - 👁 Simulator **must see** group operations made by environment
 - Required to extract
 - 🚫 Environment **must not see** what group operations simulator makes
 - Would immediately reveal that we simulate
- ⚠ First glance: **Impossible**
- 💡 Do **partial observability** via **domain separation**

Observation rules (intuition)

- 📖 Every session s gets its own group generator h_s
- 👍 Legal/unobservable:
Session s operates on h_s
- 👎 Illegal/observable:
Session s' operates on h_s

Design challenges



- 🧑 Requirements:
 - 👁 Simulator **must see** group operations made by environment
 - Required to extract
 - 👁/ Environment **must not see** what group operations simulator makes ←
 - Would immediately reveal that we simulate
- ⚠ First glance: **Impossible**
- 💡 Do **partial observability** via **domain separation**

Observation rules (intuition)

- 📜 Every session s gets its own group generator h_s
- 👍 Legal/unobservable:
Session s operates on h_s
- 👎 Illegal/observable:
Session s' operates on h_s

Design challenges



- 🧑‍🔧 Requirements:
 - 👁 Simulator **must see** ^{relevant} group operations made by environment
 - Required to extract
 - 🚫 Environment **must not see** what group operations simulator makes
 - Would immediately reveal that we simulate
- ⚠ First glance: **Impossible**
- 💡 Do **partial observability** via **domain separation**

Observation rules (intuition)

- 📜 Every session s gets its own group generator h_s
- 👍 Legal/unobservable:
Session s operates on h_s
- 👎 Illegal/observable:
Session s' operates on h_s

Simplified $\mathcal{O}' = \text{G-oGG}$

private random inj. $\tau : \mathbb{G} \rightarrow S$

public rnd generator h_s for each session s

public poly variable X_s for gen of each session s

private representation $R[e]$ for each $e \in S$, initially $R[h_s] = X_s$

op(g_1, g_2):

$s =$ caller session

$result = \tau(\tau^{-1}(g_1) + \tau^{-1}(g_2))$

$R[result] = R[g_1] + R[g_2]$ //bookkeep sum of polynomials

if $R[result] \notin \mathbb{Z}_p[X_s]$: //invalid in caller session

 Add $(g_1, g_2, result)$ to *public observation list*

return $result$

observe():

return *public observation list*



Simplified $\mathcal{O}' = \text{G-oGG}$



```
private random inj.  $\tau : \mathbb{G} \rightarrow S$ 
public rnd generator  $h_s$  for each session  $s$ 
public poly variable  $X_s$  for gen of each session  $s$ 
private representation  $R[e]$  for each  $e \in S$ , initially
 $R[h_s] = X_s$ 

 $\text{op}(g_1, g_2)$ :
 $s = \text{caller session}$ 
 $\text{result} = \tau(\tau^{-1}(g_1) + \tau^{-1}(g_2))$ 
 $R[\text{result}] = R[g_1] + R[g_2]$  //bookkeep sum of polynomials
if  $R[\text{result}] \notin \mathbb{Z}_p[X_s]$ : //invalid in caller session
    Add  $(g_1, g_2, \text{result})$  to public observation list
return  $\text{result}$ 
```

observe():
return *public observation list*

Example ops with caller session s

- $17X_s$ op $X_{s'}$ observable
- $(17X_s + 3X_{s'})$ op X_s observable
- $17X_s$ op $4X_s$ unobservable
- $(17X_s + 0X_{s'})$ op X_s unobservable

Actual G-oGG

- 🧑🧑 Multiple generators per session
- 🧑 Oblivious sampling
- 🌟 Pairing operations



Application: Groth16

Theorem 1. *Π -G16 UC-realizes $\mathcal{F}$ -wNIZK in the $\mathcal{F}$ -CRS-hybrid model in the presence of $\mathcal{G}$ -oGG. Concretely, for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}_{\text{G16}}$ such that for every $\mathcal{Z}$ that makes at most $q_{\mathcal{Z}}$ queries to $\mathcal{G}$ -oGG, $q_{\mathcal{P}}$ queries to the PROVE interface, and $q_{\mathcal{V}}$ queries to the VERIFY interface,*

$$\begin{aligned} & \left| \Pr[\text{EXEC}_{\mathcal{F}\text{-wNIZK}, \mathcal{Z}, \mathcal{S}_{\text{G16}}, \mathcal{G}\text{-oGG}}(\lambda, z) = 1] - \Pr[\text{EXEC}_{\Pi\text{-G16}, \mathcal{Z}, \mathcal{A}, \mathcal{G}\text{-oGG}}(\lambda, z) = 1] \right| \\ & \leq 72 \cdot d \cdot (m + d + q_{\mathcal{Z}} + (m + d)q_{\mathcal{P}} + \ell q_{\mathcal{V}} + 1)^2 / (p - 1) \end{aligned}$$

and $\mathcal{S}_{\text{G16}}$ performs in total the following operations:

- at most $3q_{\mathcal{P}} + 9q_{\mathcal{V}} + 2q_{\mathcal{Z}} + 3d + m + 8$ queries to $\mathcal{G}$ -oGG
 - at most $(2\ell + 8)q_{\mathcal{P}} + (3q_{\mathcal{Z}} + 2\ell + 2)q_{\mathcal{V}} + (d + 1)(3m + 11)$ field operations
- where d, m, ℓ depend on the circuit size (see Functionality 6).

Groth16 proof challenges



Groth16 proof challenges

Superpower

Observe group operations to
extract witness

Extraction



Groth16 proof challenges

Superpower

Observe group operations to
extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

Extraction



Groth16 proof challenges

Superpower

Observe group operations to
extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

Solution

Argue that valid proofs cannot
contain foreign generators



Groth16 proof challenges

Superpower

Observe group operations to
extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

💡 Solution

Argue that valid proofs cannot
contain foreign generators



Groth16 proof challenges

Superpower

Observe group operations to extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

Solution

Argue that valid proofs cannot contain foreign generators

Extraction



Simulation



Superpower

Use CRS trapdoor to generate proofs without witness



Groth16 proof challenges

Superpower

Observe group operations to extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

Solution

Argue that valid proofs cannot contain foreign generators

Extraction



Simulation



Superpower

Use CRS trapdoor to generate proofs without witness

Challenge

Prover/Simulator GGM ops must not be observable



Groth16 proof challenges

Superpower

Observe group operations to extract witness

Challenge

Cannot observe *everything*
(only *my* session's generator(s))

Solution

Argue that valid proofs cannot contain foreign generators

Extraction



Simulation



Superpower

Use CRS trapdoor to generate proofs without witness

Challenge

Prover/Simulator GGM ops must not be observable

Solution

Prover/simulator only operates on CRS elements



Summary

- 🕶 UC is hard, English is too malleable
 - Hypothesis: **Pseudocode** helps
- 🚧 The benefits of Pseudocode are not obvious, we build **case studies**:
 - zkSNARKs, Blockchain Bridges, Threshold Signatures
- 🧐 zkSNARK's simulator/extractor inherently **need superpower**.
Makes composition difficult.
- 👁 Modeled **restricted observable global generic group UC functionality G-oGG**.
 - Approach: Partial observability based on **domain separation**.
- 🤖 Proved **Groth16 SNARK UC-secure** in presence of G-oGG with zero overhead!





Backup

ROM vs GGM

Local ROM: bad model 😞 Both sessions use SHA-3, why am I getting different hashes?	Local GGM: bad model 😞 Both sessions use BLS12-381, why are elements incompatible?
<i>Global</i> ROM: lose observability. Remodel. Environment/other protocols can access global ROM without going through the simulator.	<i>Global</i> GGM: lose observability. Remodel. Environment/other protocols can access global GGM without going through the simulator.
Domain separation: $RO(s, x')$ is “valid/in-session” iff caller is in session s. Invalid queries are observable.	Domain separation: $op(g_1, g_2)$ is “valid/in-session” iff g_1, g_2 are based on caller session’s generator h_s
ZK: honest parties only make “valid” unobservable queries within their domain. PoK: when environment / protocol in session $s' \neq s$ queries related to domain s, it’s observable.	



$\mathcal{F}_{\text{NIZK}}$

$\mathcal{F}_{\text{NIZK}}$

INIT_{sid}()

1: $T \leftarrow []$

PROVE_{sid}(x, w)

2: **if** $(x, w) \notin \mathcal{R}$ **then return** $\perp$

3: $\pi \leftarrow \mathcal{S}.\text{SIMULATE}_{\text{sid}}(x)$

4: $T \leftarrow T \cup (x, \pi)$

5: **return** π

VERIFY_{sid}(x, π)

6: **if** $(x, \pi) \in T$ **then return** 1

7: $w \leftarrow \mathcal{S}.\text{EXTRACT}_{\text{sid}}(x, \pi)$

8: **if** $(x, w) \in \mathcal{R}$ **then** $T \leftarrow T \cup (x, \pi)$

9: **if** $(x, \pi) \in T$ **then**

10: **return** 1

11: **else**

12: **return** 0

$\mathcal{F}_{\text{wNIZK}}$

$\mathcal{F}_{\text{wNIZK}}$

INIT_{sid}()

1: $T \leftarrow []$

PROVE_{sid}(x, w)

2: **if** $(x, w) \notin \mathcal{R}$ **then return** $\perp$

3: $\pi \leftarrow \mathcal{S}.\text{SIMULATE}_{\text{sid}}(x)$

4: $T \leftarrow T \cup (x, \pi)$

5: **return** π

VERIFY_{sid}(x, π)

6: **if** $(x, \pi) \in T$ **then return** 1

7: $w \leftarrow \mathcal{S}.\text{EXTRACT}_{\text{sid}}(x, \pi)$

8: **if** $(x, w) \in \mathcal{R}$ **then** $T \leftarrow T \cup (x, \pi)$

9: **if** $(w = \text{maul} \wedge (x, *) \in T)$ **then** $T \leftarrow T \cup (x, \pi)$

10: **if** $(x, \pi) \in T$ **then**

11: **return** 1

12: **else**

13: **return** 0