Cryptography on the Blockchain

Vassilis Zikas

RPI

IACR Summer School on Blockchain Techs

Aggelos Kiayias, Hong-Shen Zhou, and Vassilis Zikas, Fair and Robust Multi-Party Computation using a Global Transaction Ledger, EUROCRYPT 2016.

Bitcoin

Bitcoin

What is bitcoin and how does it work?	

Bitcoin

What is bitcoin and how does it work?	
Is it secure?	(in restricted models)

Bitcoin

What is bitcoin and how does it work?	✔
Is it secure?	(in restricted models) ✔ ?
What do we get from it?	

Bitcoin

What is bitcoin and how does it work?	✓
Is it secure?	(in restricted models) ✓ ?
What do we get from it?	

What Crypto can get from Bitcoin?

What Crypto can get from Bitcoin?

In this talk
"Bitcoin = Ledger-based cryptocurrency"

What Crypto can get from Bitcoin?

In this talk
"Bitcoin = Ledger-based cryptocurrency"

A public transaction ledger

Some economic stuff …

What Crypto can get from Bitcoin?

In this talk
"Bitcoin = Ledger-based cryptocurrency"

A public transaction ledger

A bulletin board with a filter on what gets written there

Some economic stuff …

What Crypto can get from Bitcoin?

In this talk
"Bitcoin = Ledger-based cryptocurrency"

A public transaction ledger

A bulletin board with a filter on what gets written there

Some economic stuff …

People (good or bad) want money

The Public Transaction Ledger

“What is *exactly* the problem that bitcoin solves?”

AK, 2016

The Public Transaction Ledger

"What is *exactly* the problem that bitcoin solves?"

AK, 2016

The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions

The Public Transaction Ledger

"What is *exactly* the problem that bitcoin solves?"
AK, 2016

"Backbone" [GarayKiayiasLeonardos15]

The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions

The Public Transaction Ledger

"What is *exactly* the problem that bitcoin solves?"
AK, 2016

"Backbone" [GarayKiayiasLeonardos15]

The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions

This goal can be captured as an ideal Transaction-Ledger Functionality

The Public Transaction Ledger

"What is *exactly* the problem that bitcoin solves?"
AK, 2016

"Backbone" [GarayKiayiasLeonardos15]

The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions

This goal can be captured as an ideal Transaction-Ledger Functionality

"If we had a trusted third party instead of the Bitcoin network, how would we expect it to behave?"

Crypto On Blockchain

Outline

- The functionality offered by blockchains
- Leveraging Security Loss with Coins

  … in Secure Function Evaluation (SFE)
- A formal cryptographic (UC) model for security proofs

Crypto On Blockchain

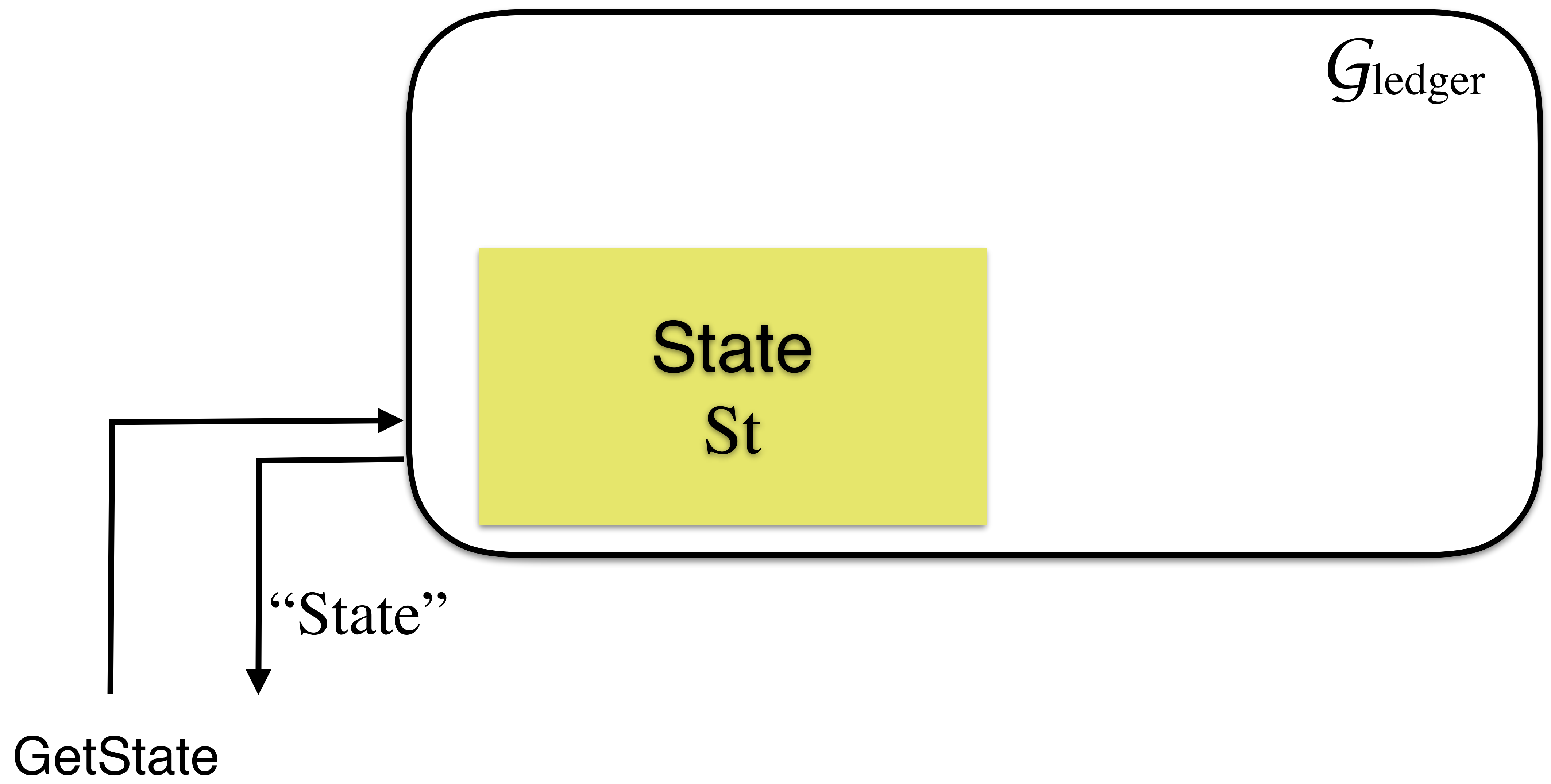
Outline

- The functionality offered by blockchains
- Leveraging Security Loss with Coins

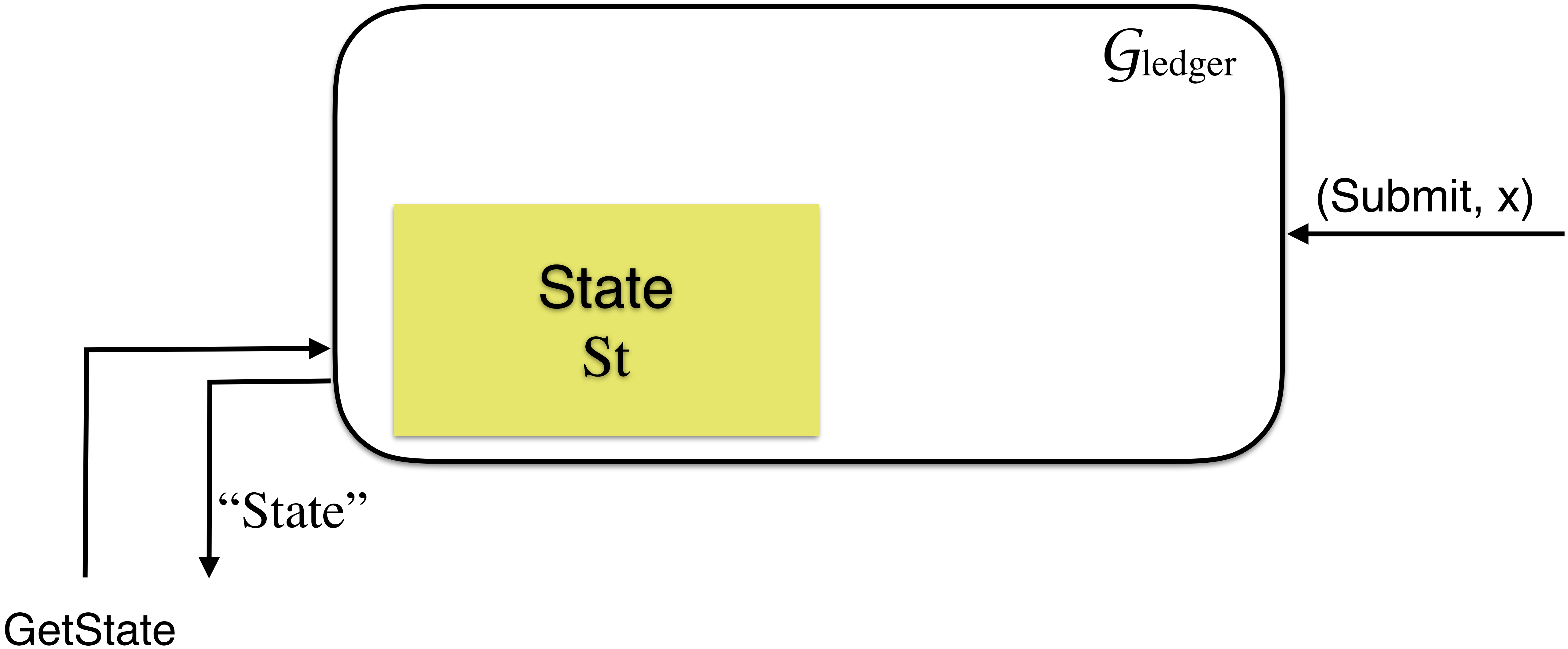
  … in Secure Function Evaluation (SFE)
- A formal cryptographic (UC) model for security proofs

The Public Transaction Ledger

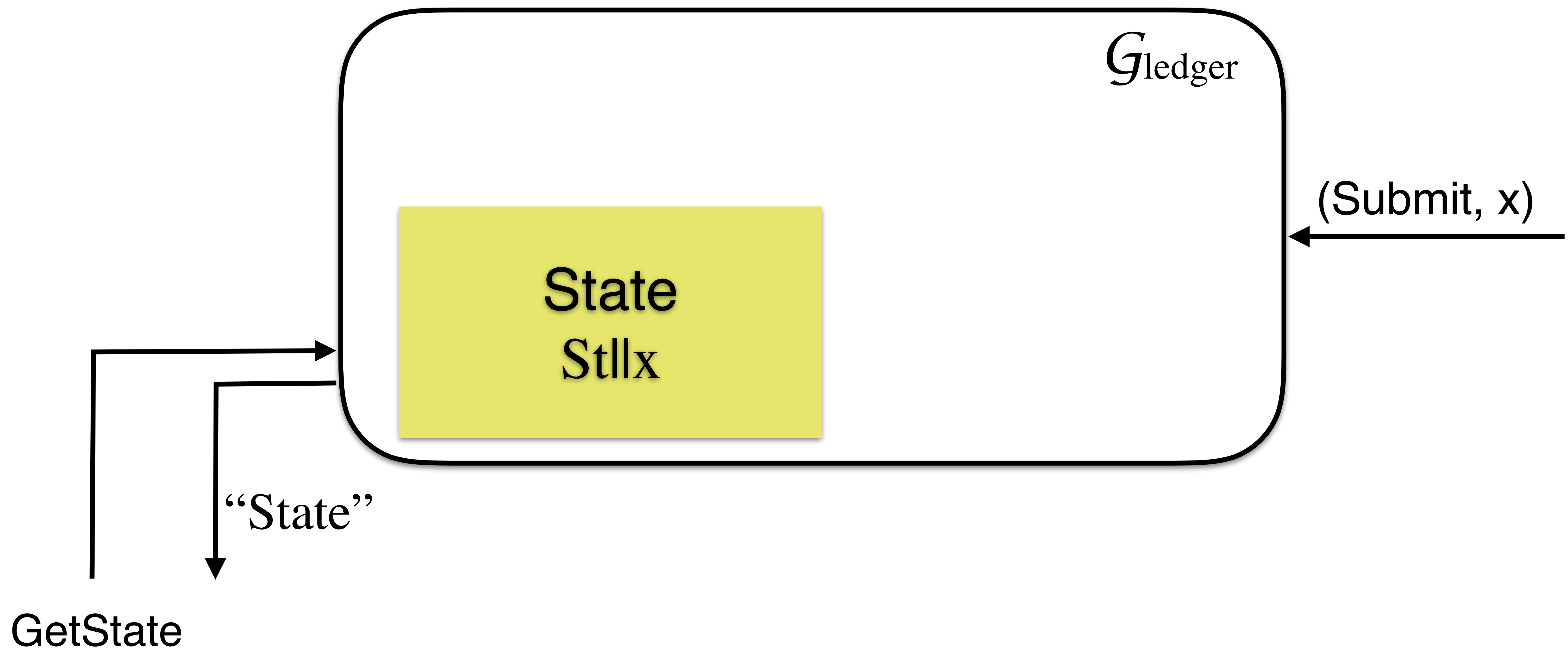
The Public Transaction Ledger



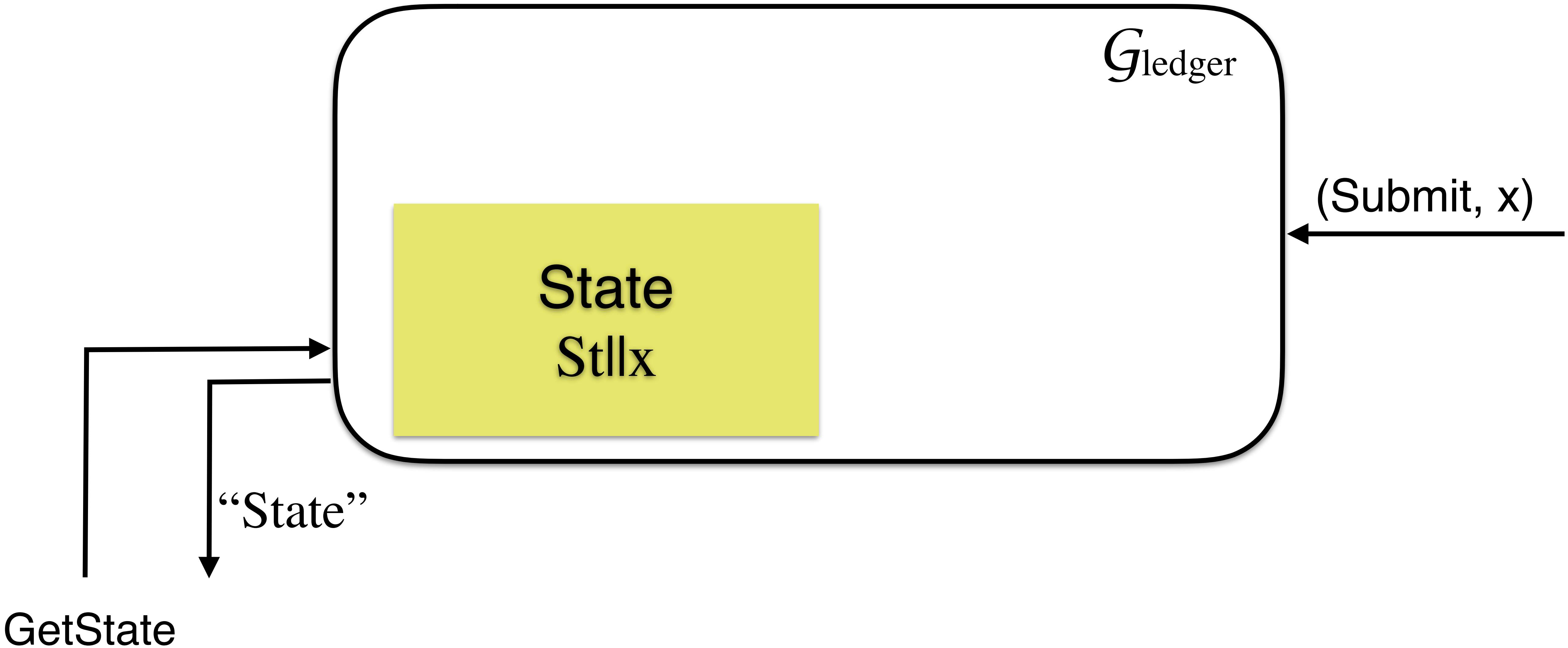
The Public Transaction Ledger



The Public Transaction Ledger

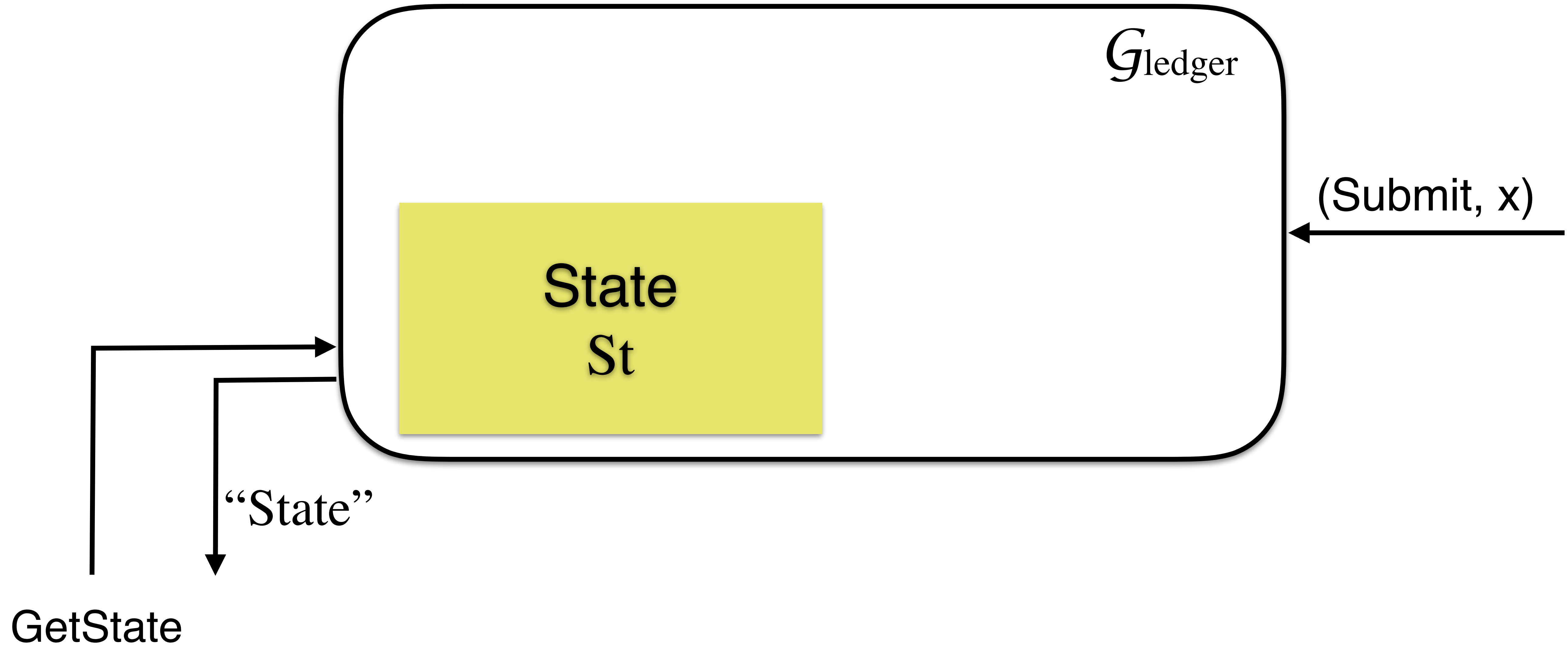


The Public Transaction Ledger



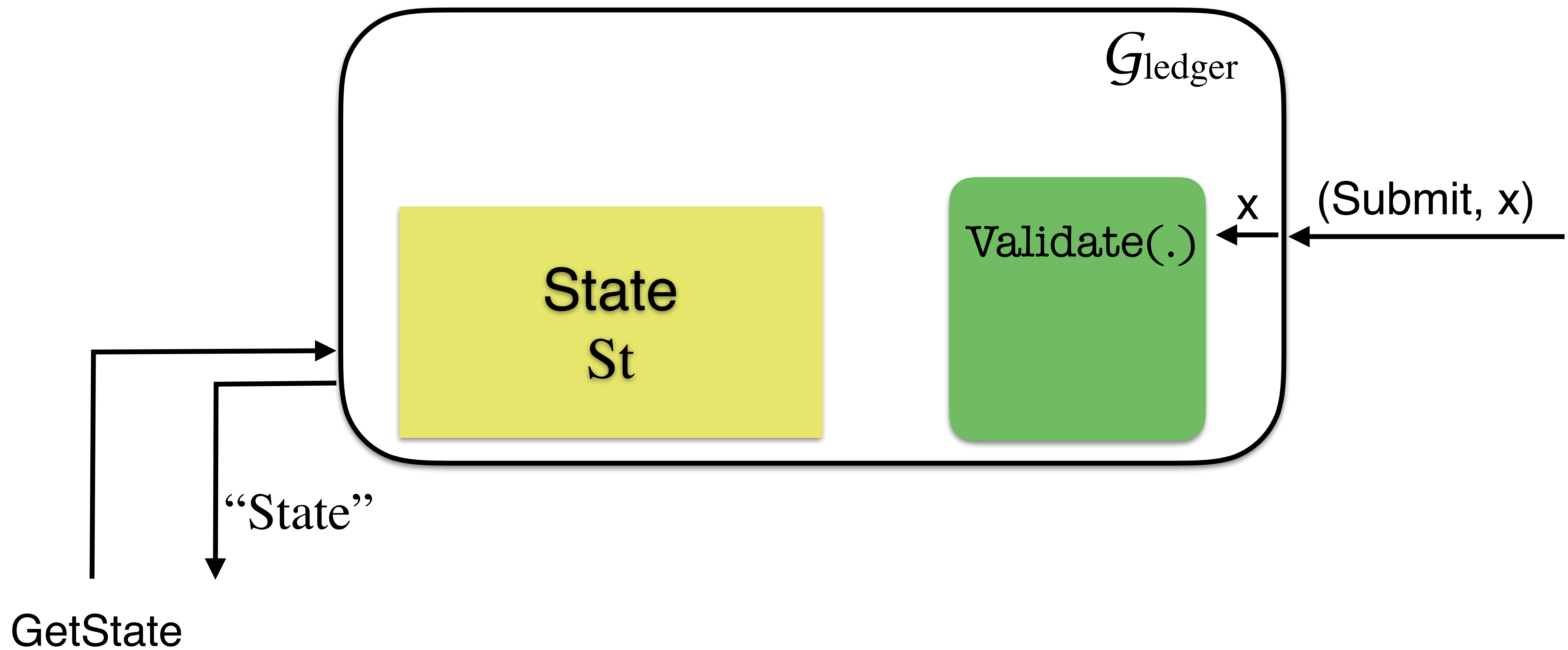
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



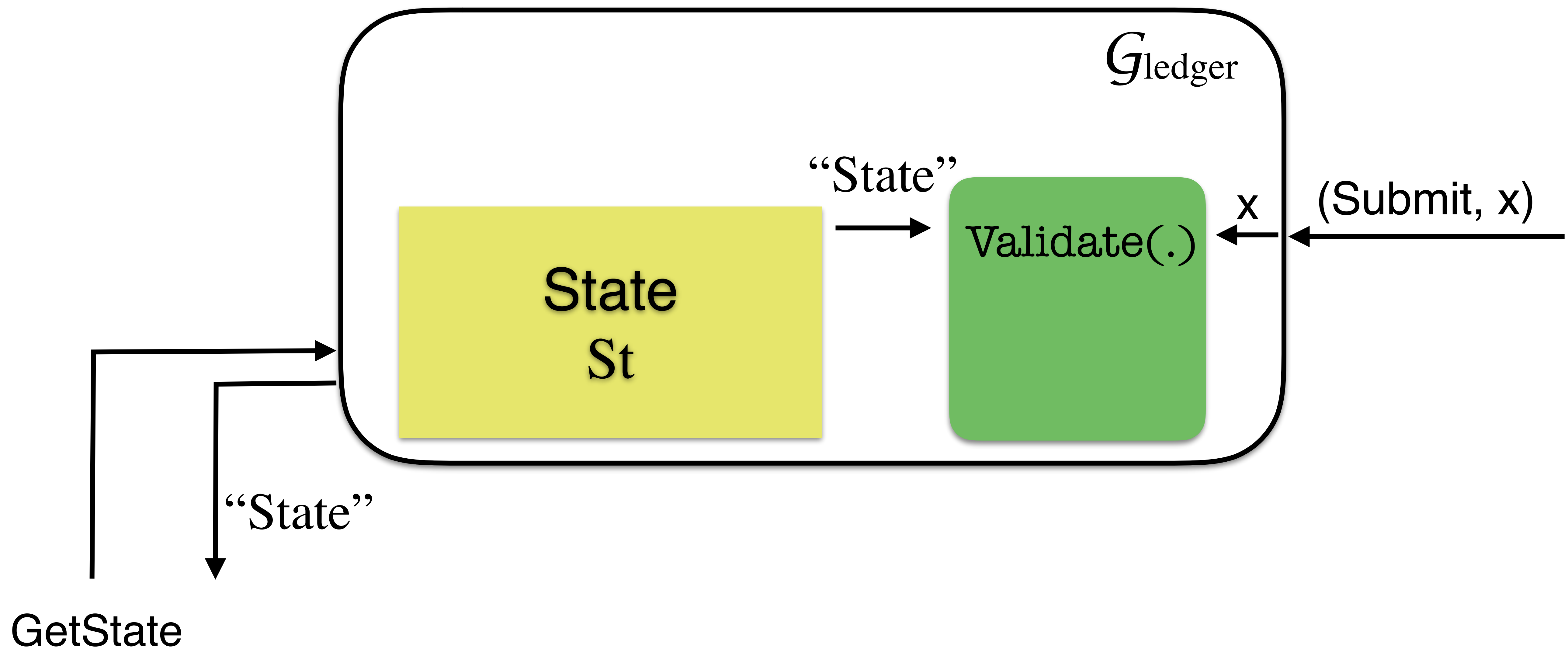
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



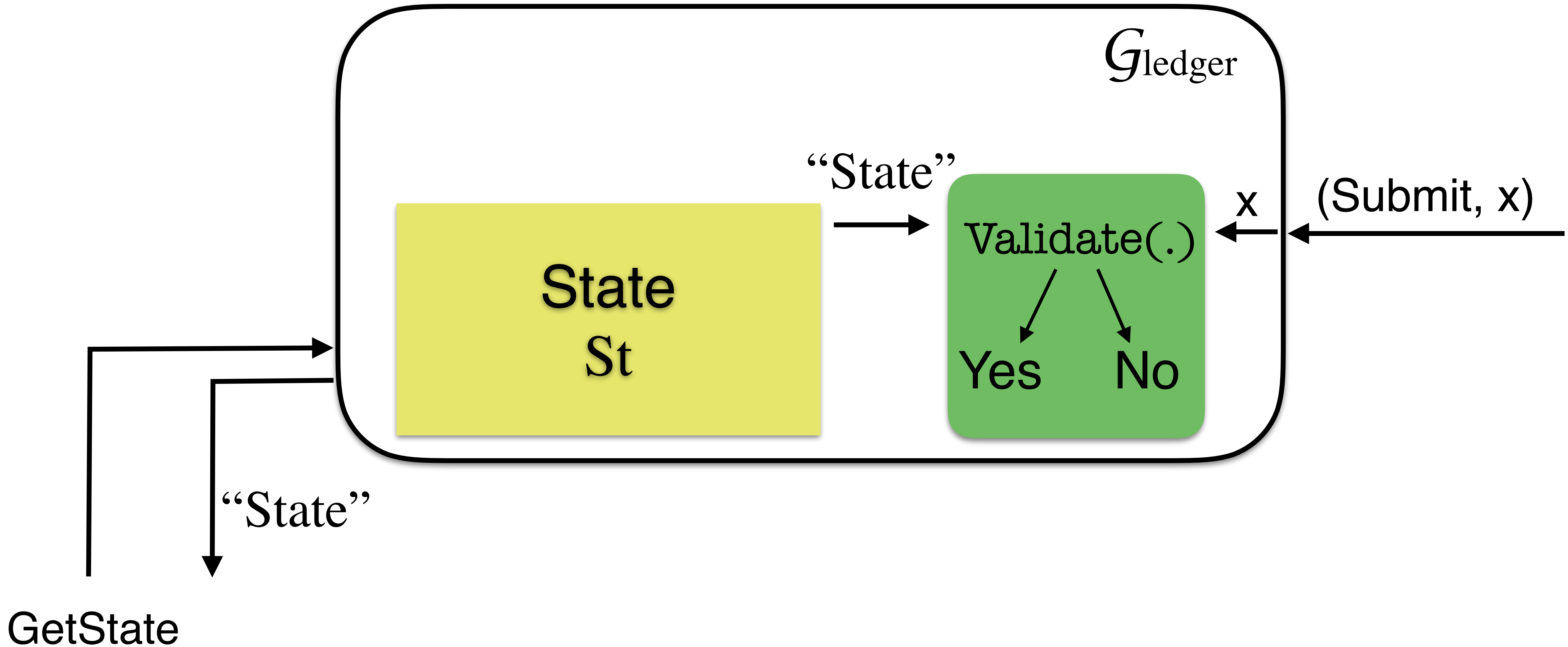
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



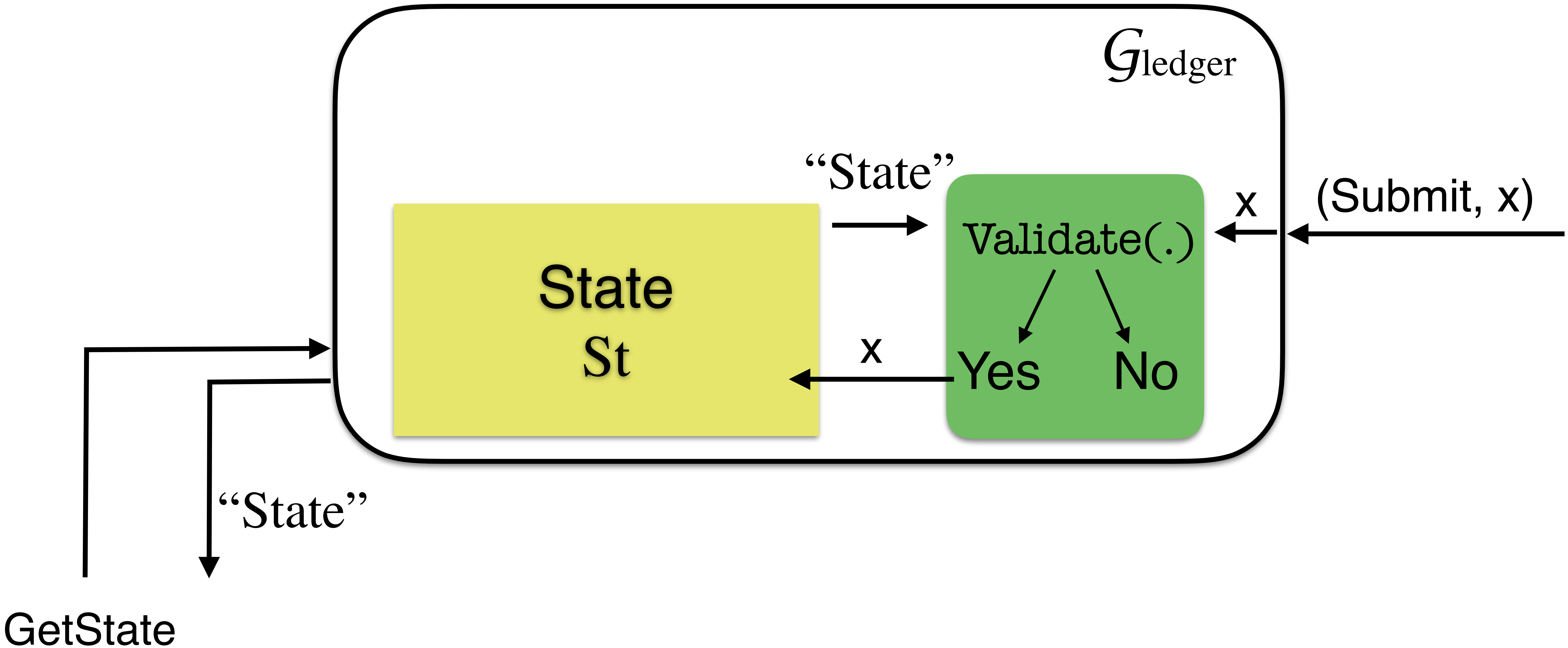
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



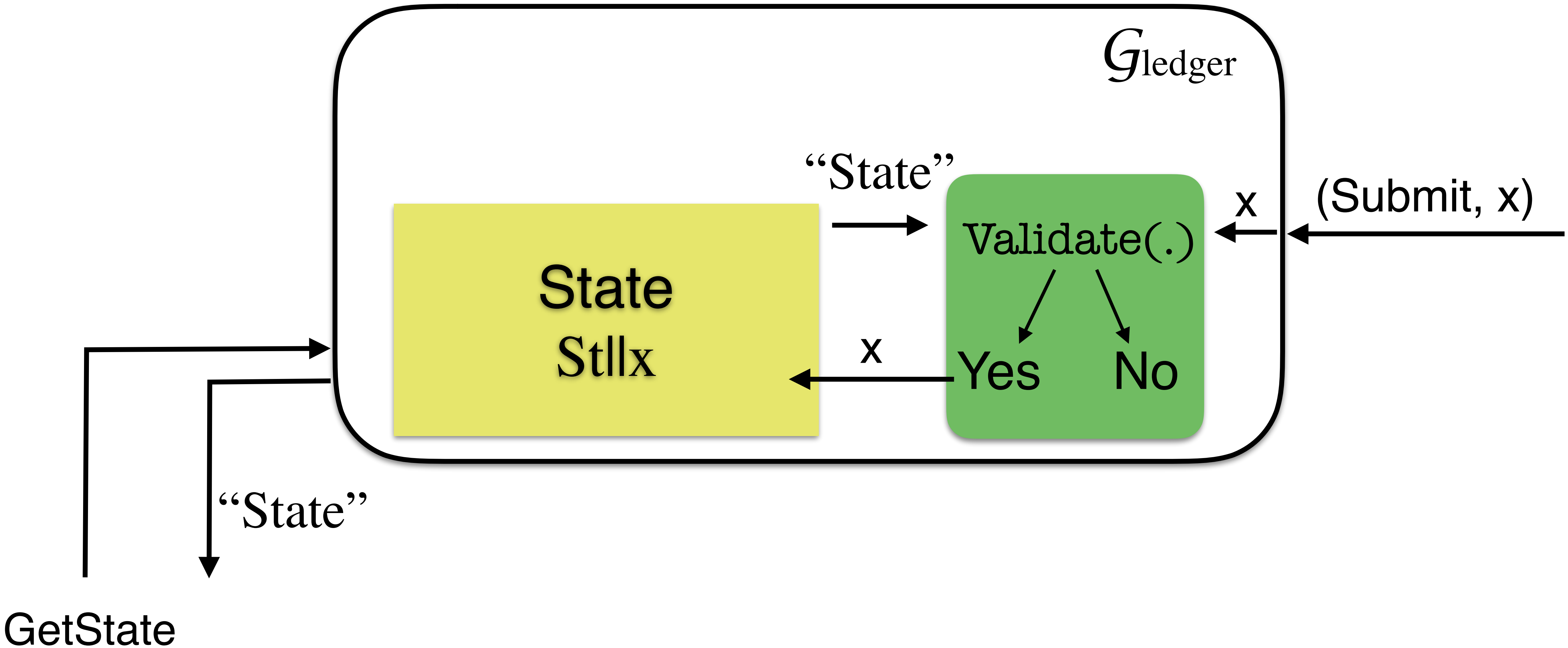
- In reality: Not a Bulletin Board
    - Inputs (transactions) are filtered

The Public Transaction Ledger



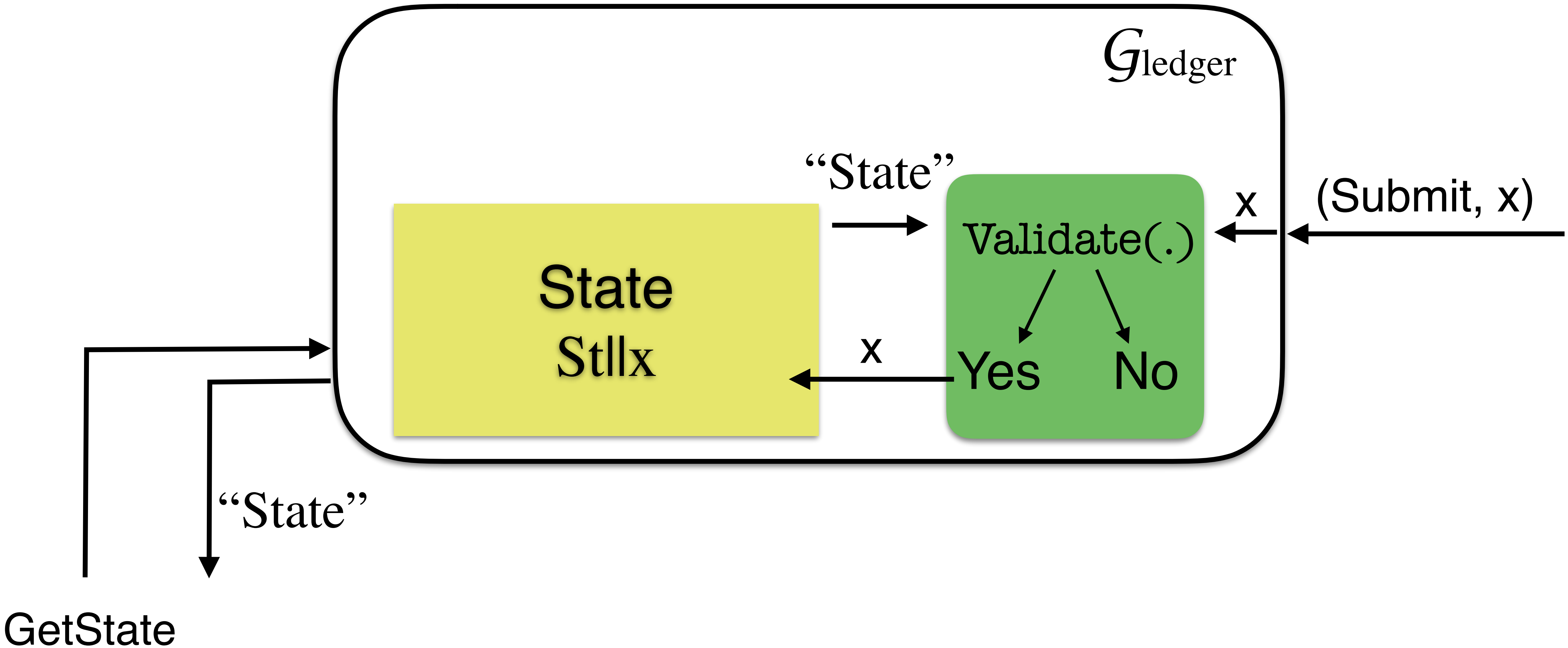
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



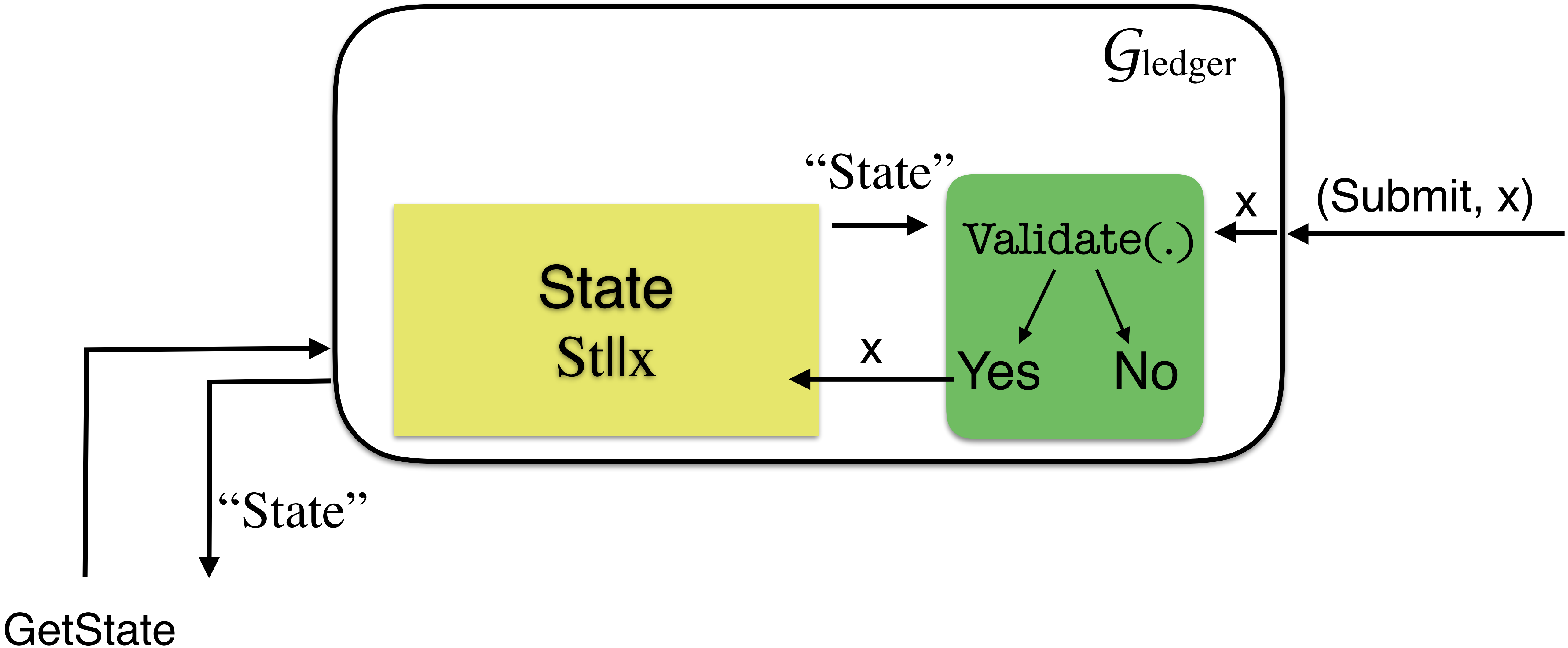
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered

The Public Transaction Ledger



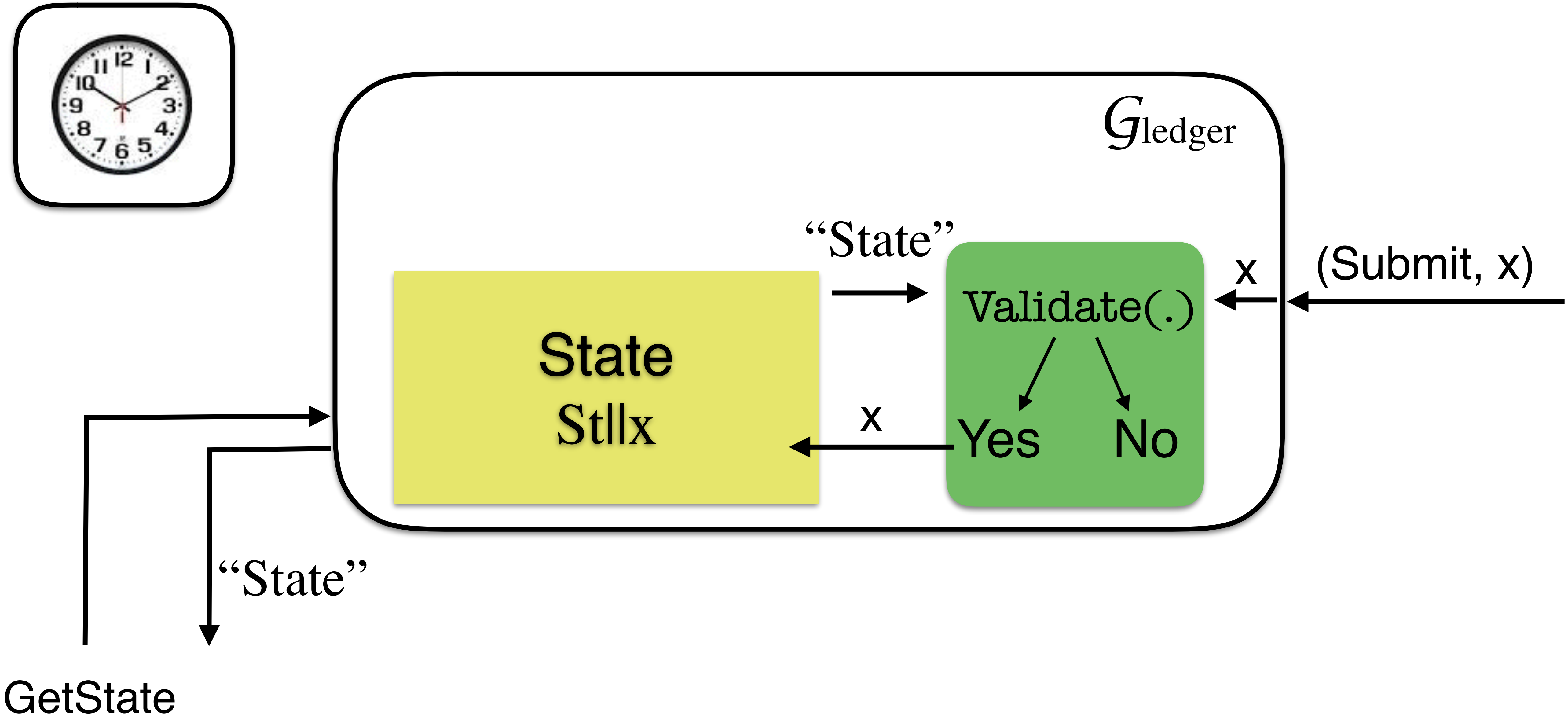
- In reality: Not a Bulletin Board
  - Inputs (transactions) are filtered
  - The order in which transactions in “State” are inserted might be adversarial … but not too adversarial

The Public Transaction Ledger



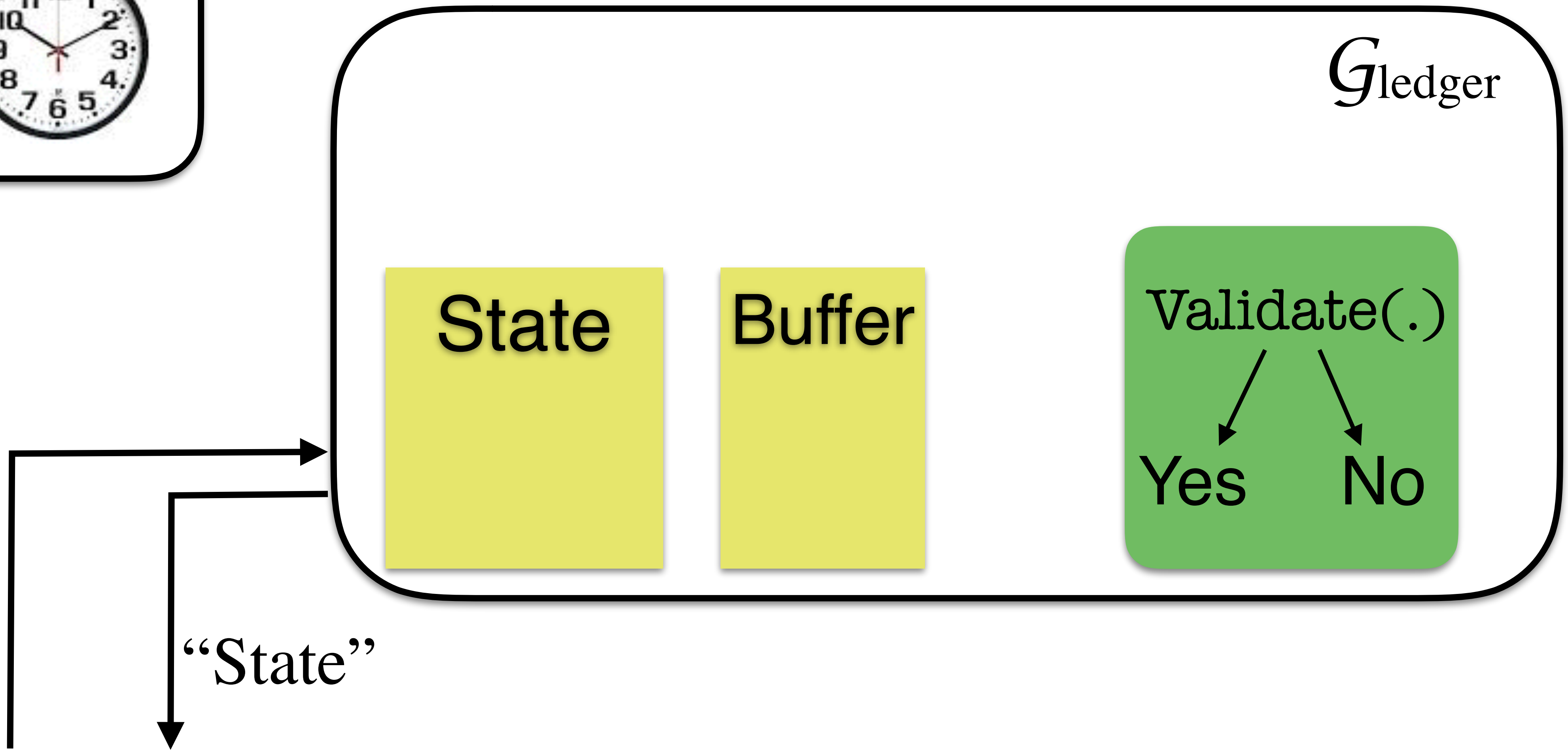
Can reorder **the recently inserted** transactions

The Public Transaction Ledger



Can reorder **the recently inserted** transactions

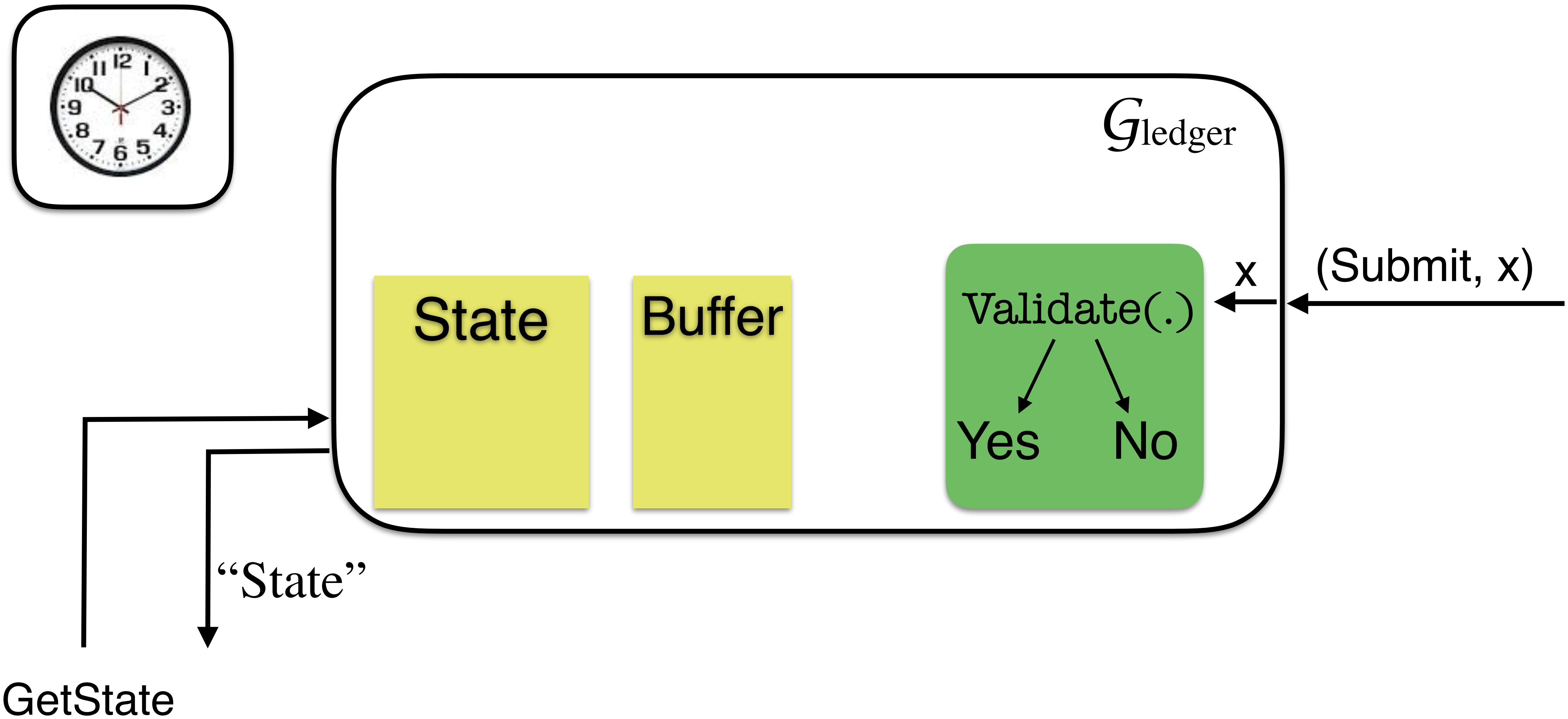
The Public Transaction Ledger & Time



GetState

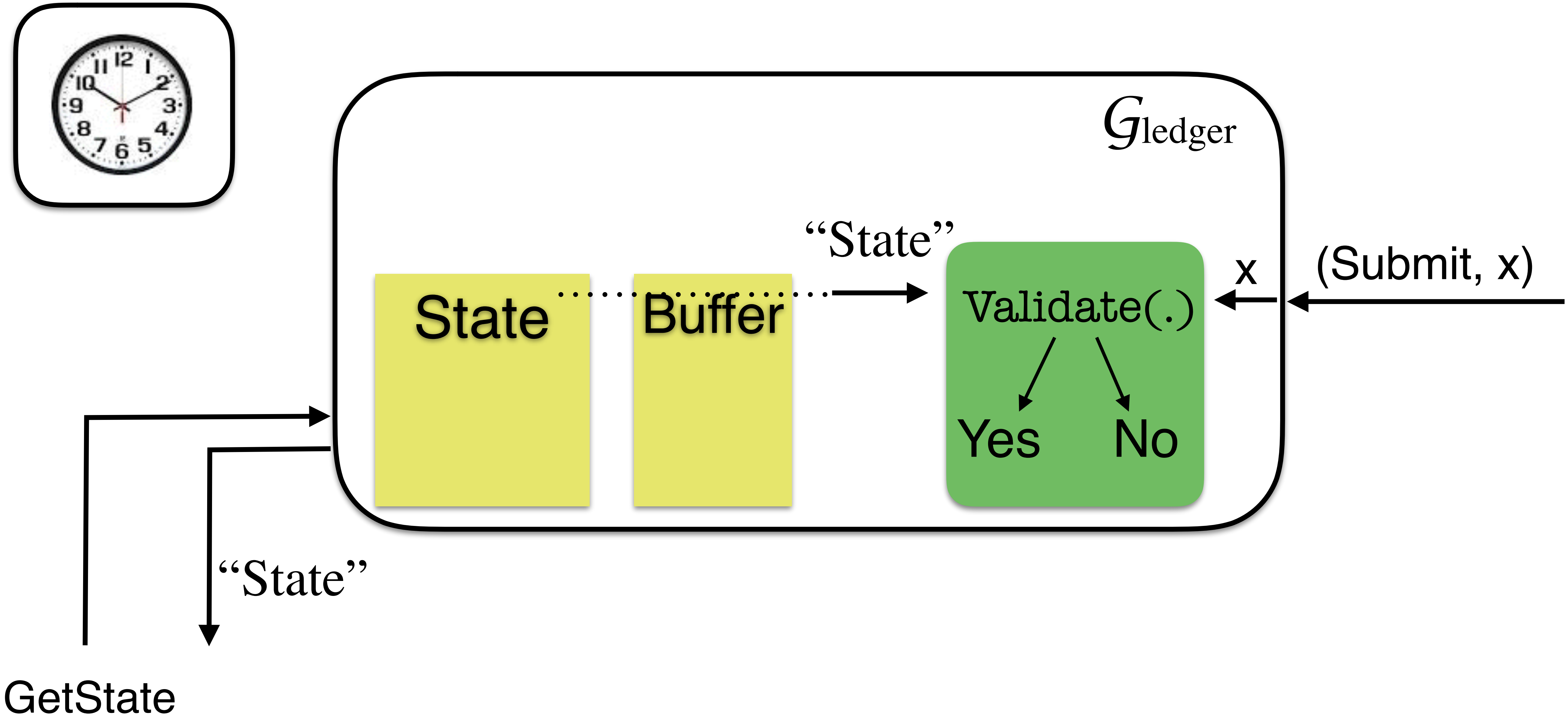
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



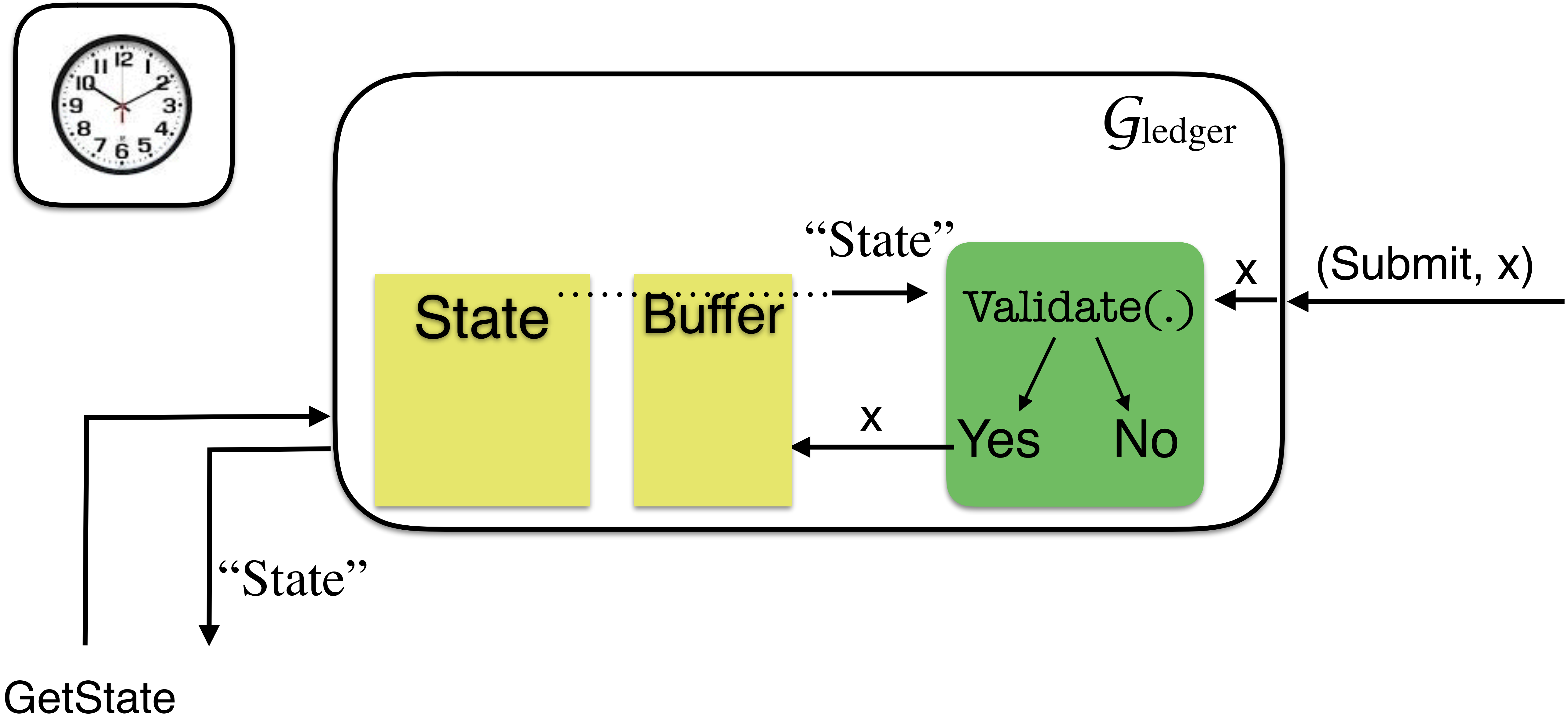
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



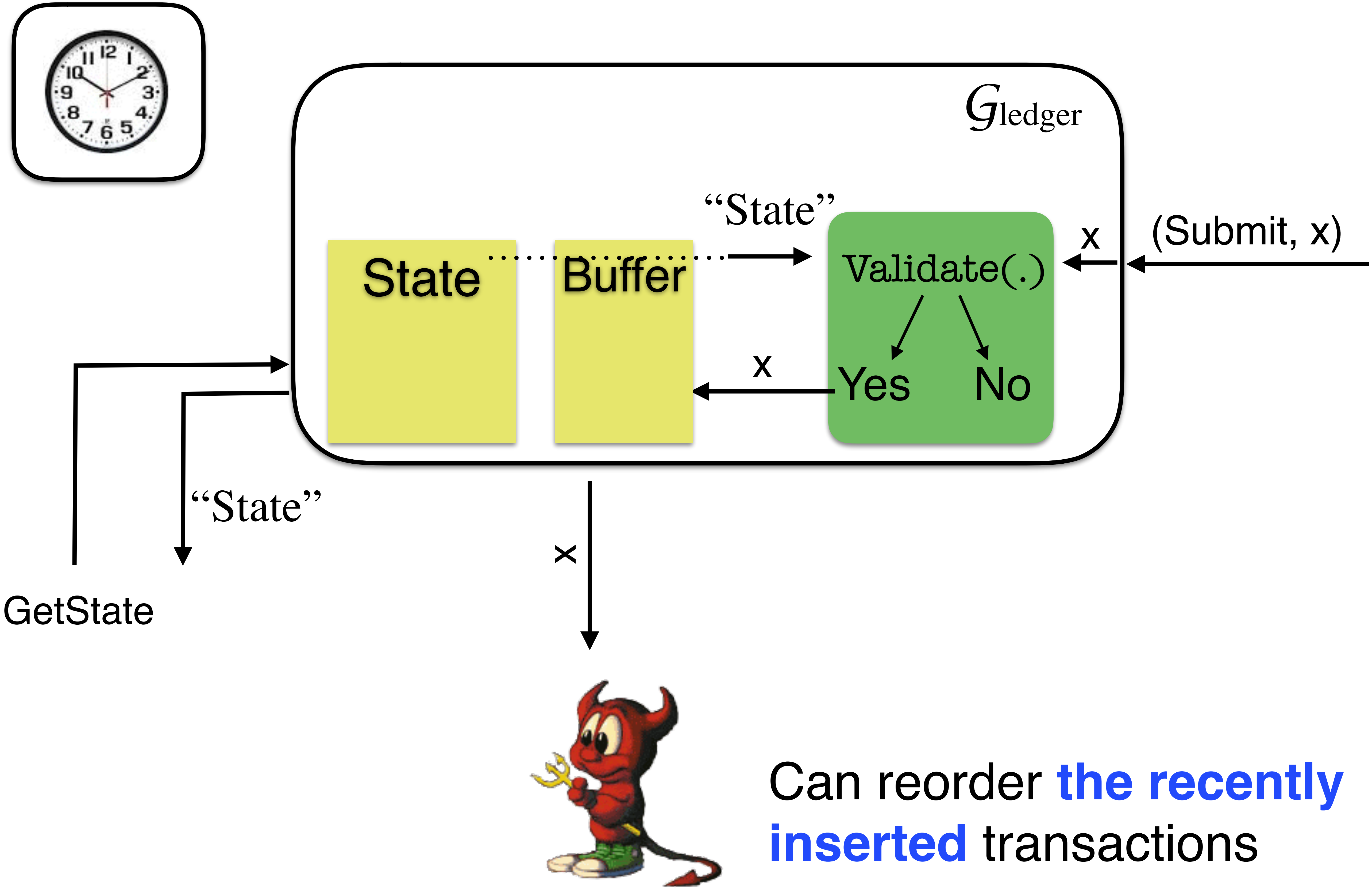
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



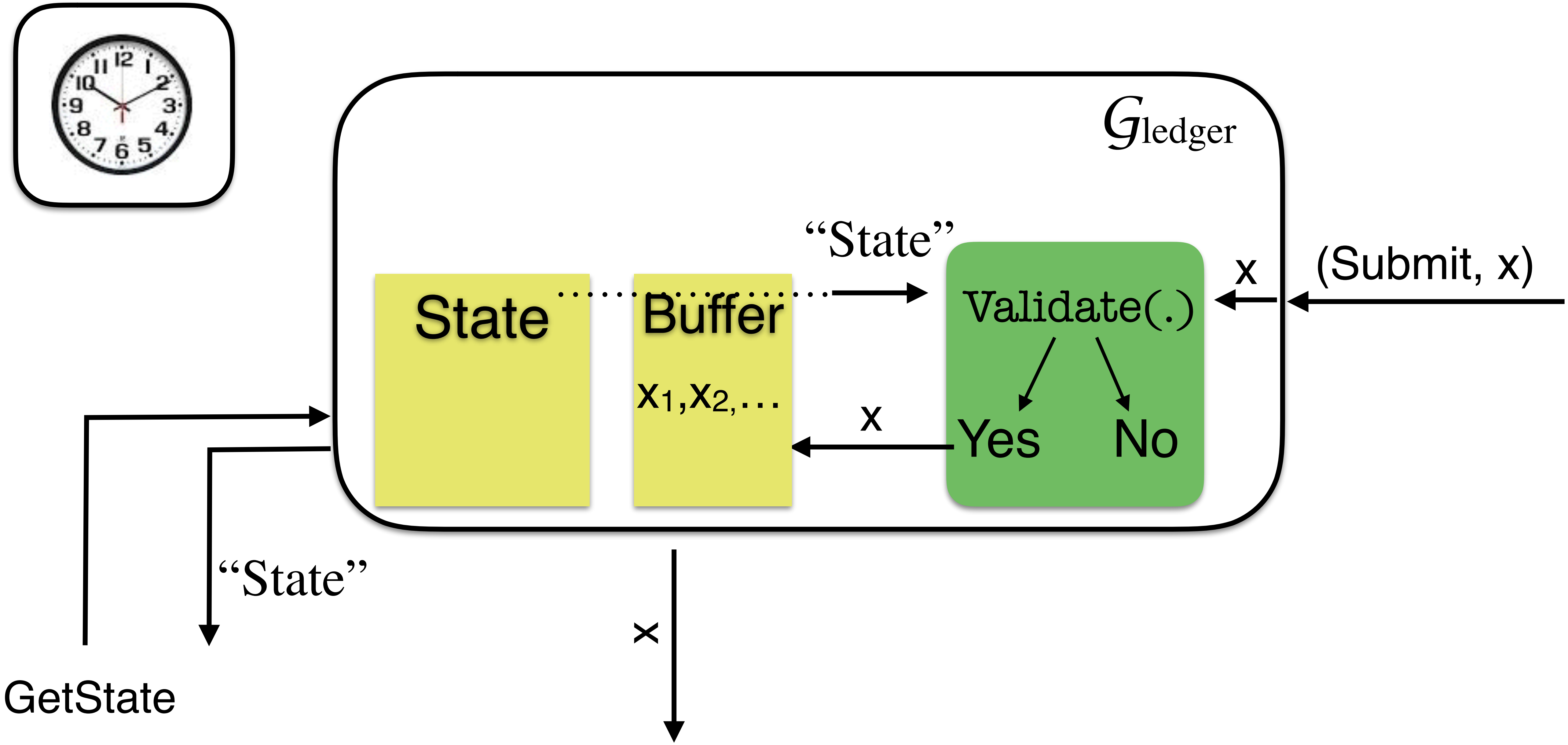
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



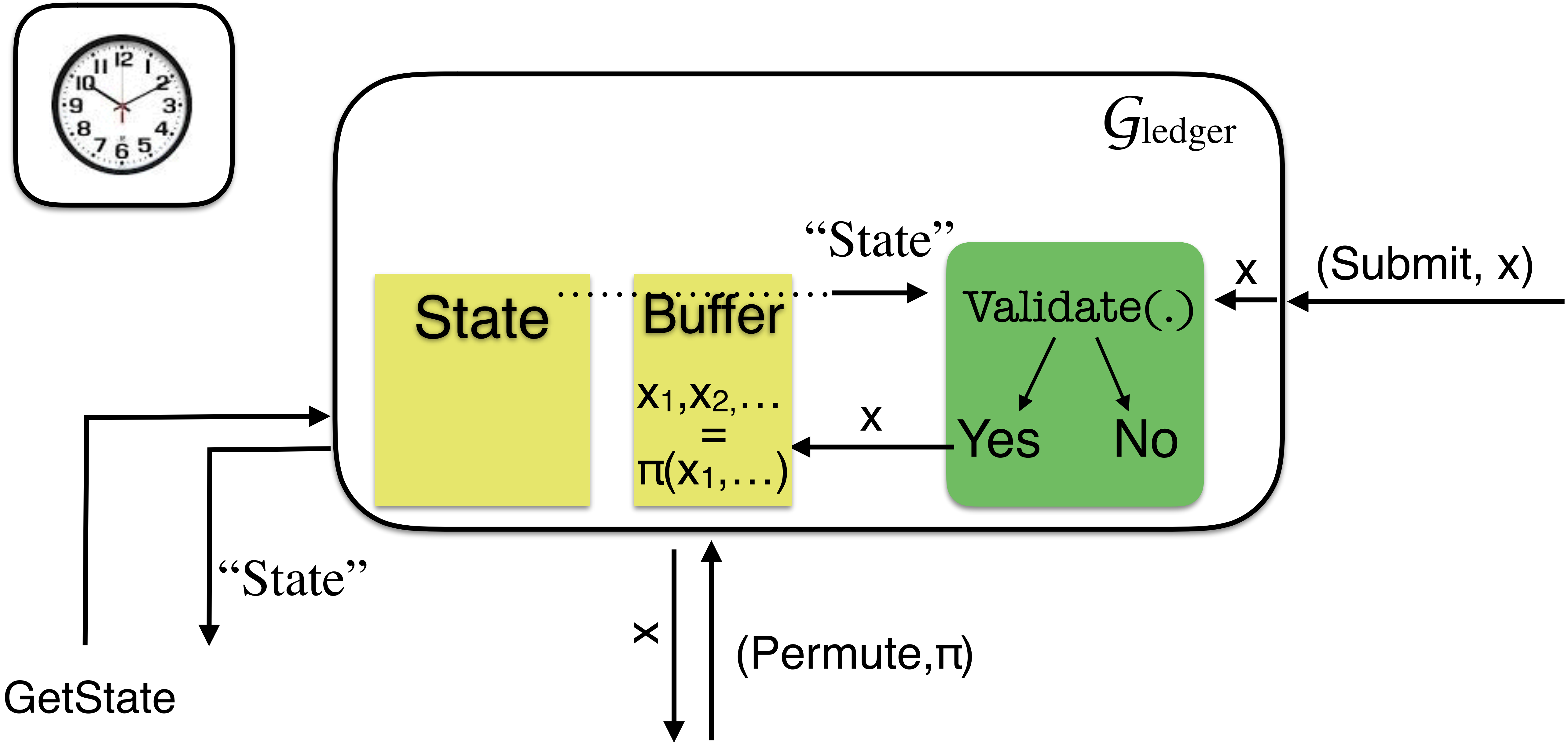
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



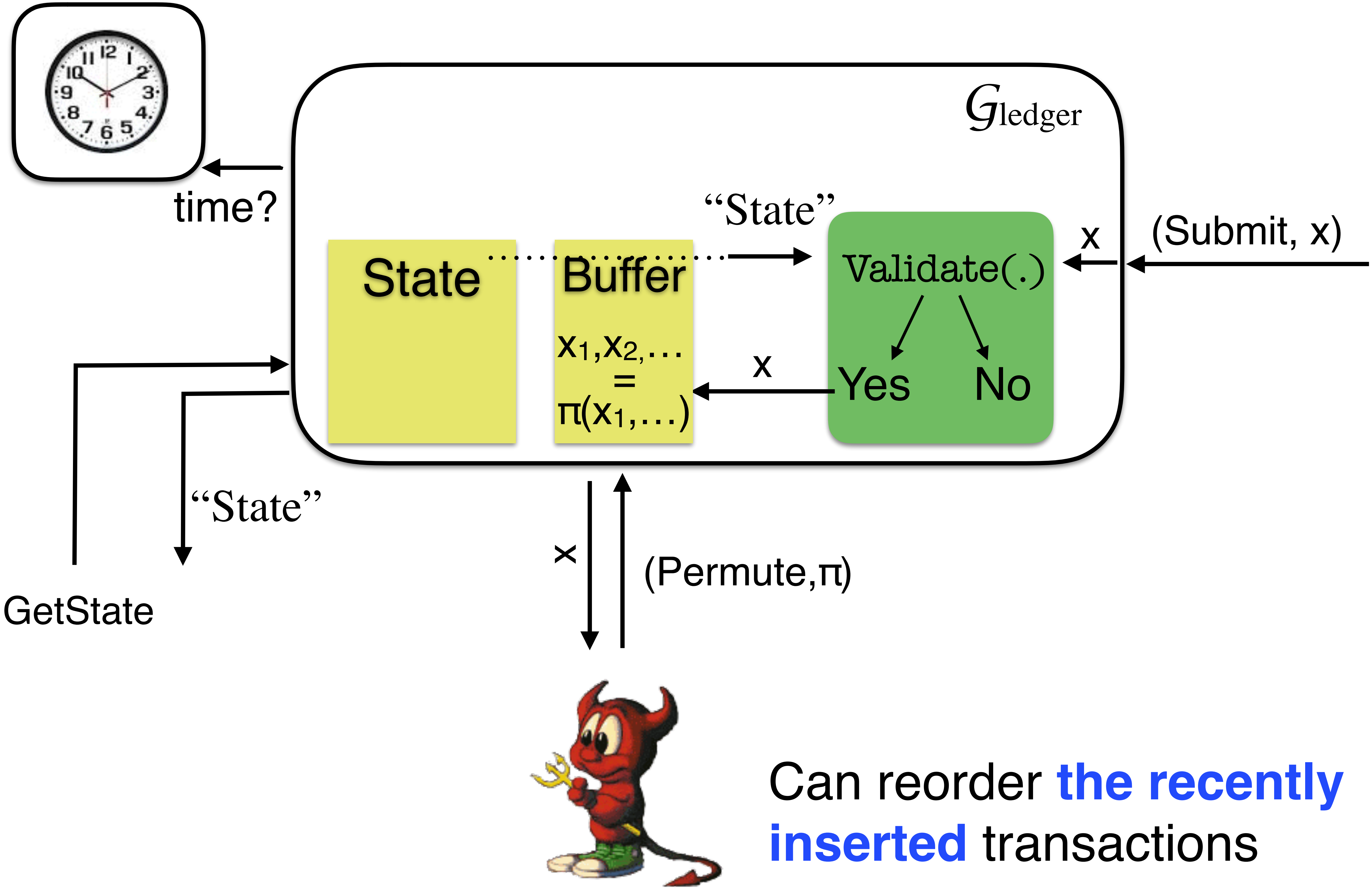
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



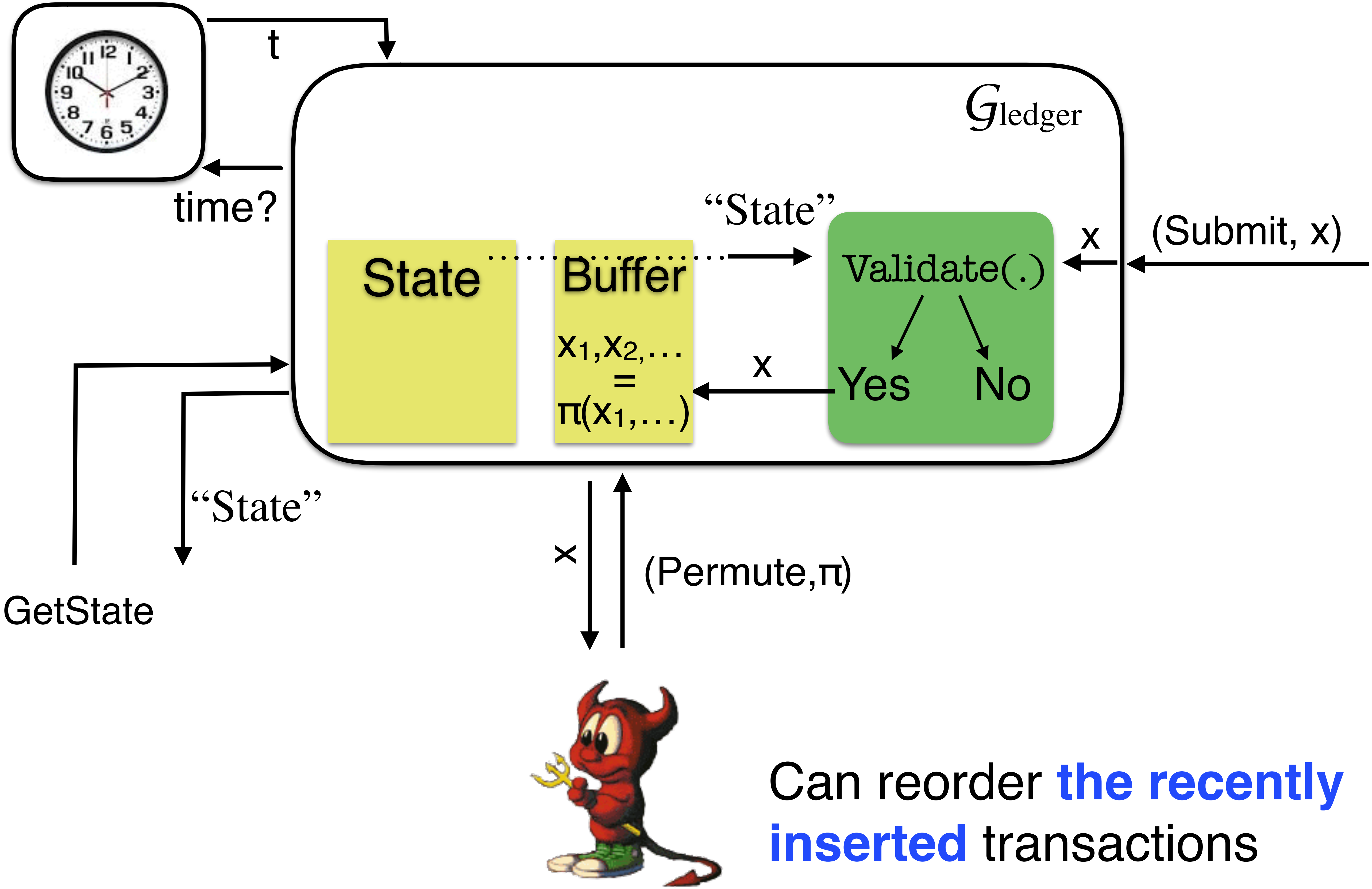
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



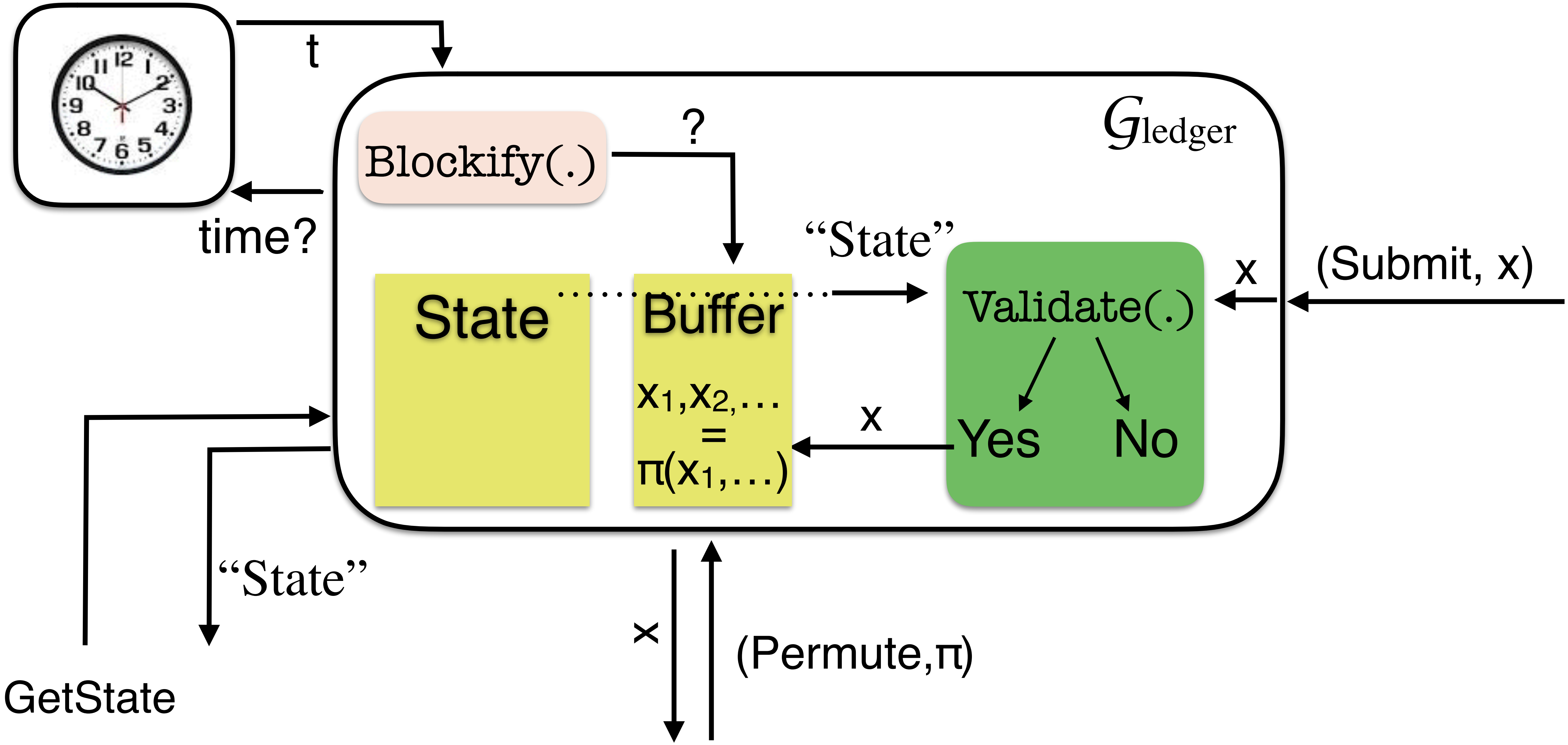
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



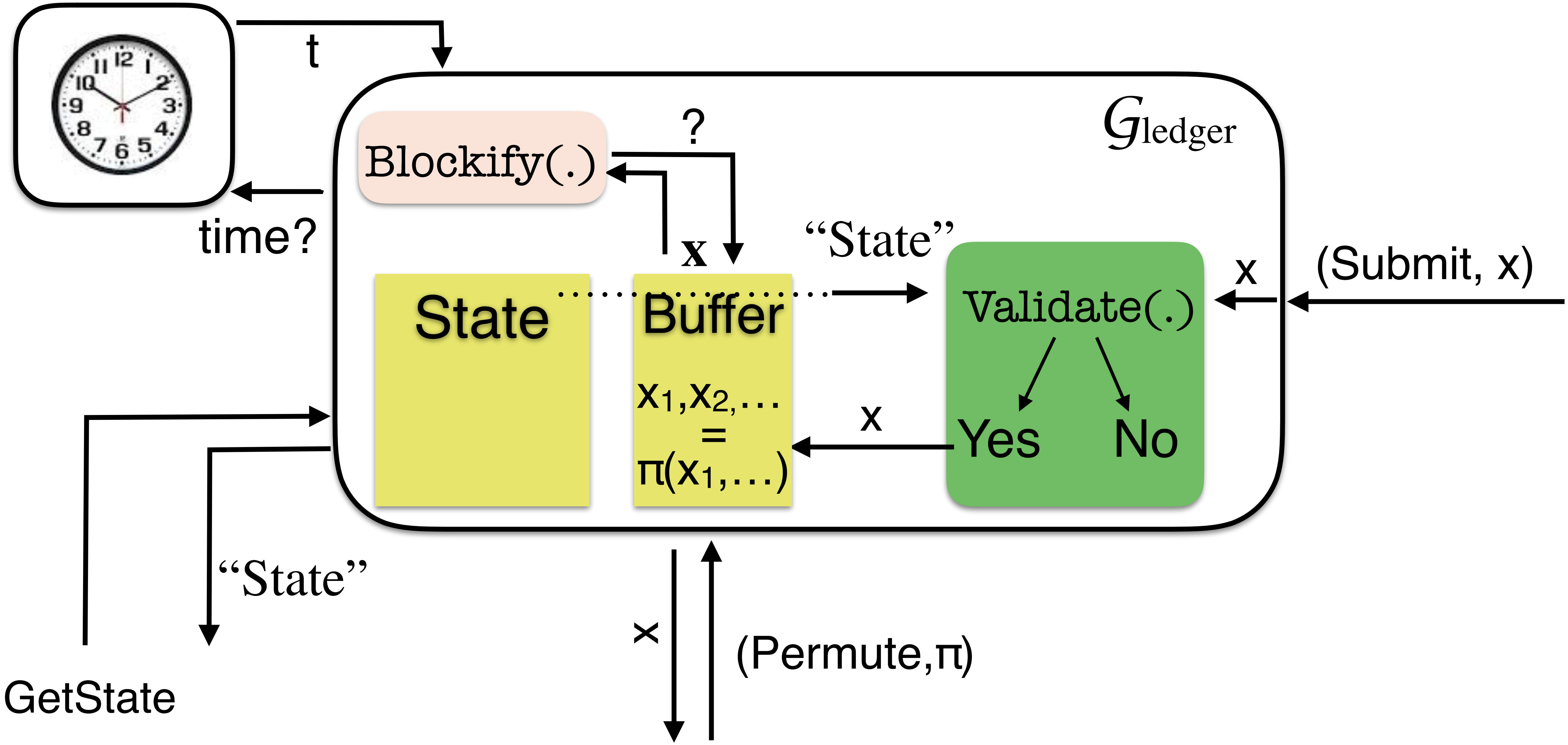
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



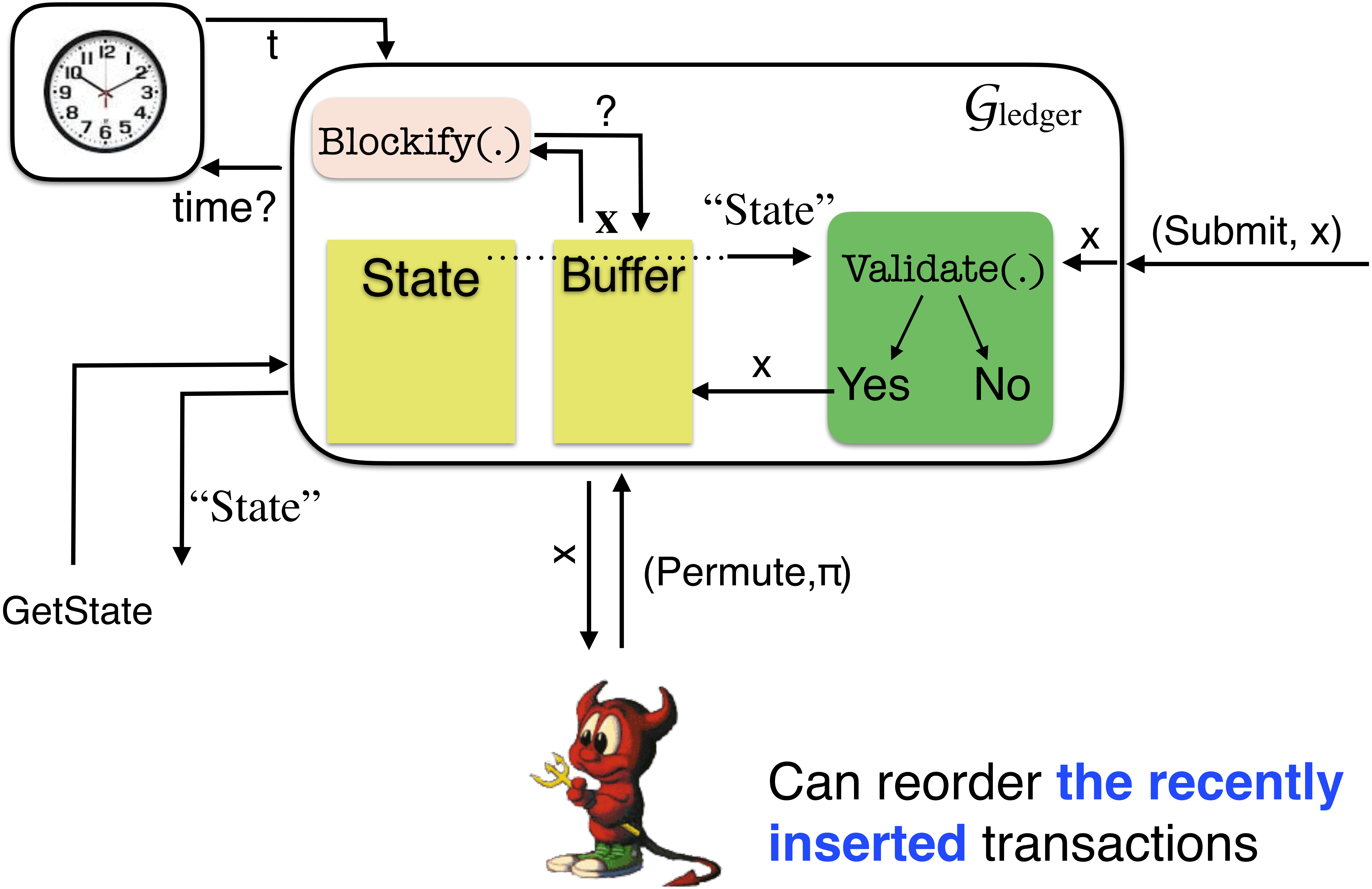
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



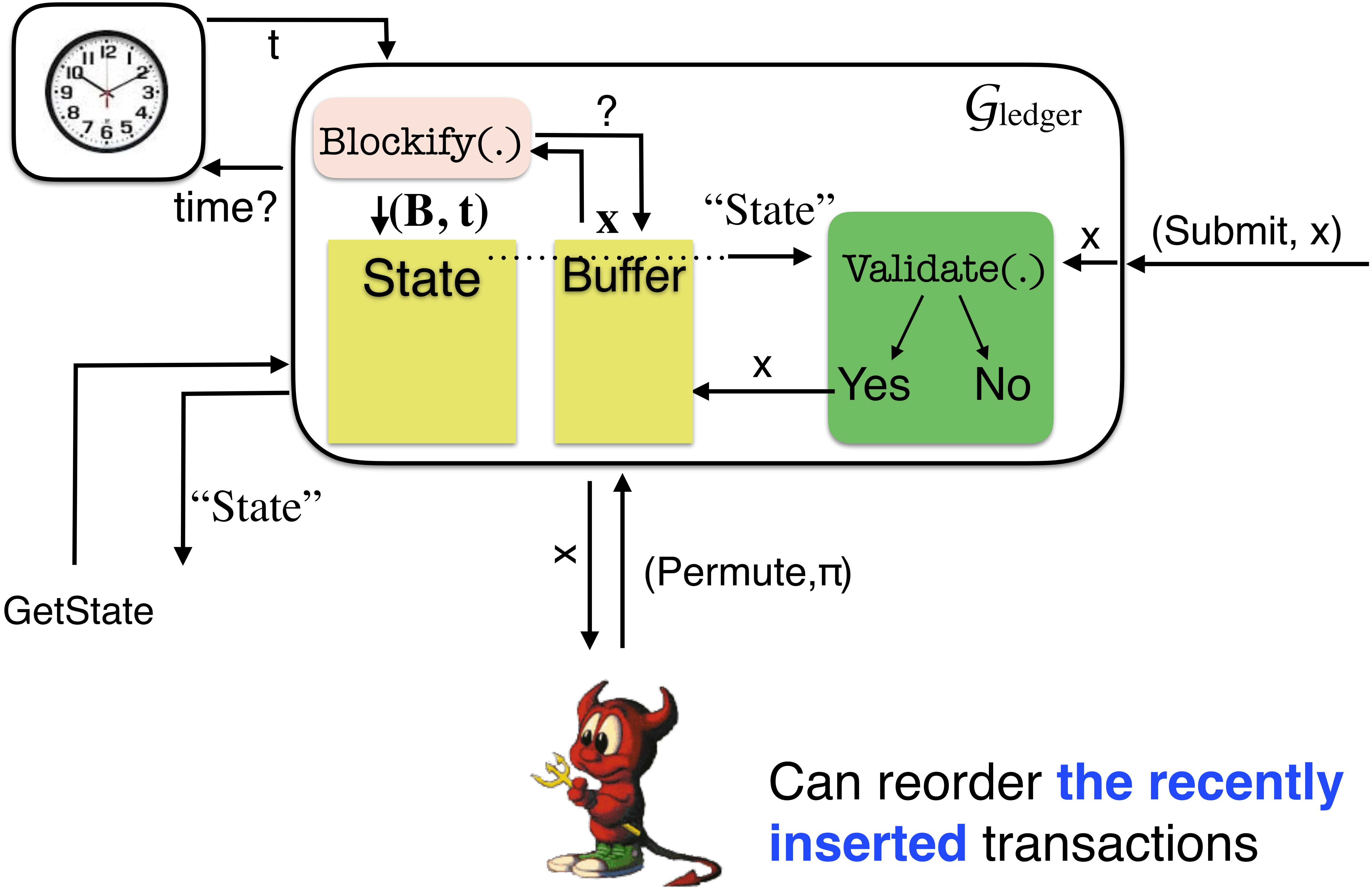
Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



Can reorder **the recently inserted** transactions

The Public Transaction Ledger & Time



Can reorder **the recently inserted** transactions

What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

$(\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})$-hybrid

(G)UC protocols

Some economic stuff …

People (good or bad) want money

What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

$(\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})$-hybrid

(G)UC protocols

- Compatibility with standard crypto-protocols (+ **composition** theorem)

Some economic stuff …

People (good or bad) want money

What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

$(\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})$-hybrid

(G)UC protocols

- Compatibility with standard crypto-protocols (+ **composition** theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast

Some economic stuff …

People (good or bad) want money

What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

$(\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})$-hybrid

(G)UC protocols

Some economic stuff …

People (good or bad) want money

"This cryptography has been around for a long time" JB 2016

- Compatibility with standard crypto-protocols (+ **composition** theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast

What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

$(\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})$-hybrid

(G)UC protocols

Some economic stuff …

People (good or bad) want money

"This cryptography has been around for a long time" JB 2016

- Compatibility with standard crypto-protocols (+ **composition** theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast

Crypto On Blockchain

Outline

- The functionality offered by blockchains
- Leveraging Security Loss with Coins

  … in Secure Function Evaluation (SFE)
- A formal cryptographic (UC) model for security proofs

Crypto On Blockchain

Outline

- The functionality offered by blockchains
- Leveraging Security Loss with Coins

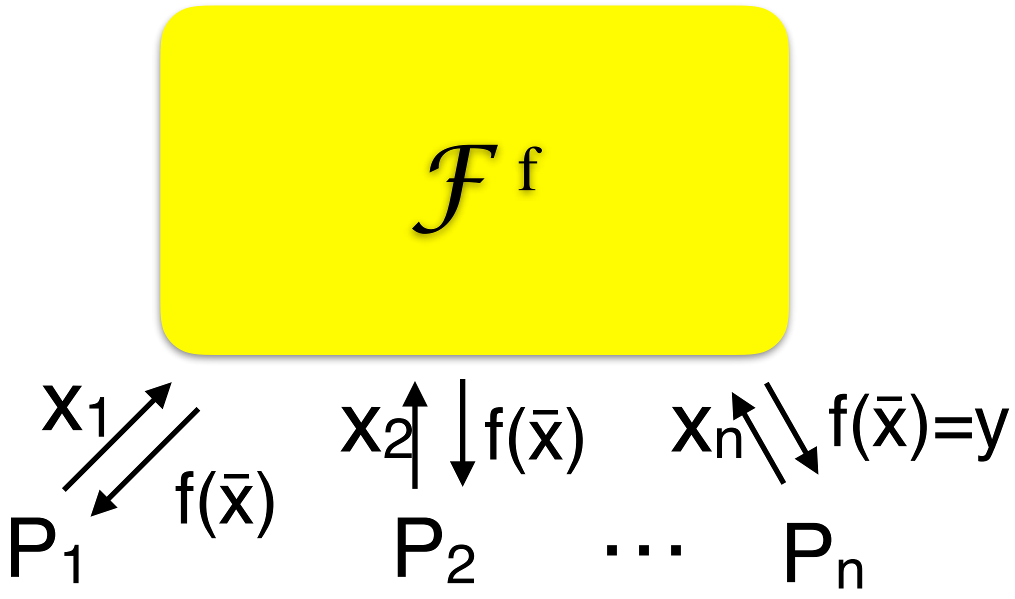
  … in Secure Function Evaluation (SFE)
- A formal cryptographic (UC) model for security proofs

Secure Function Evaluation (SFE)

Goal: Parties $P_1,\ldots,P_n$ with inputs $x_1,\ldots,x_n$ wish to compute a function $f(x_1,\ldots,x_n)$ *securely*

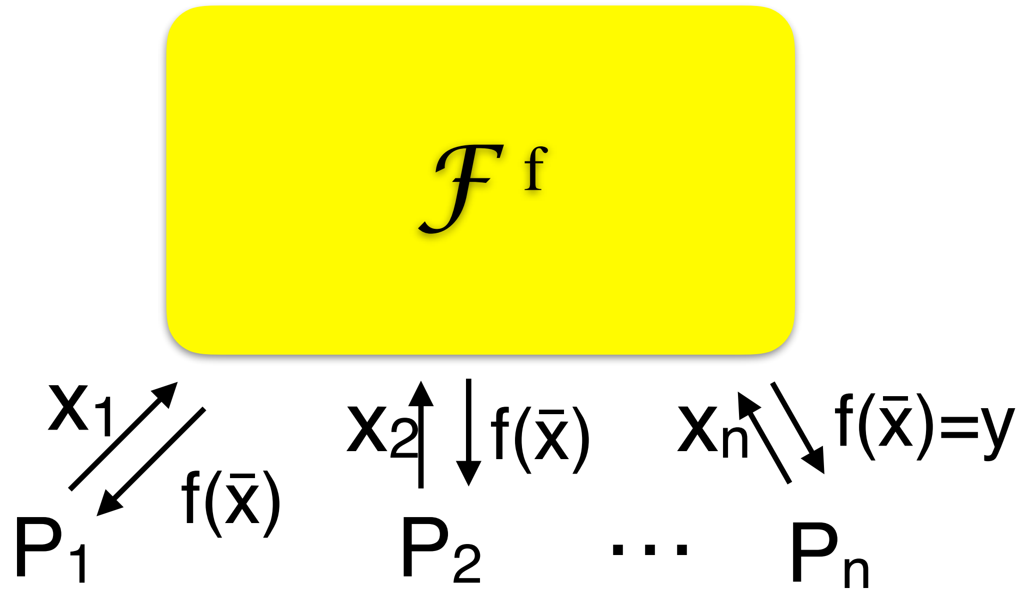
Secure Function Evaluation (SFE)

Ideal World

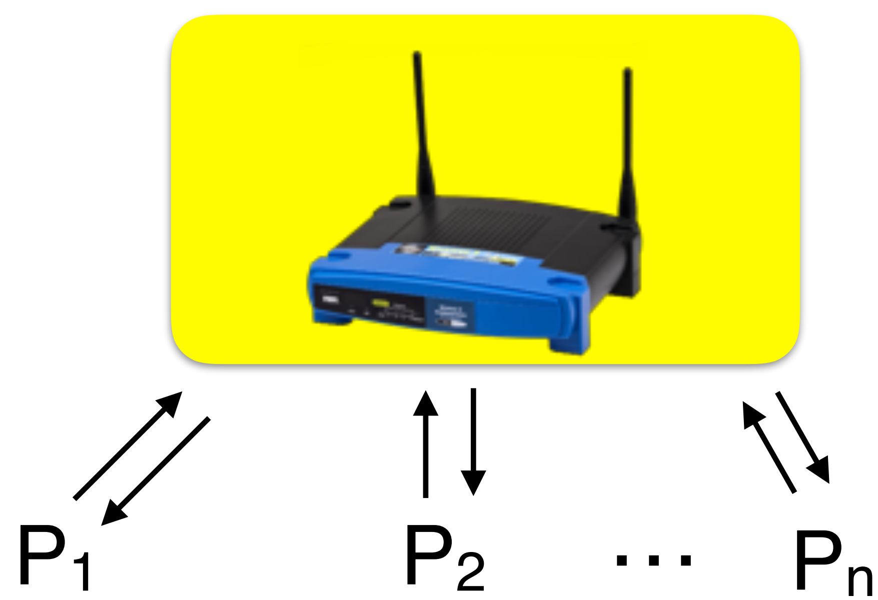


Secure Function Evaluation (SFE)

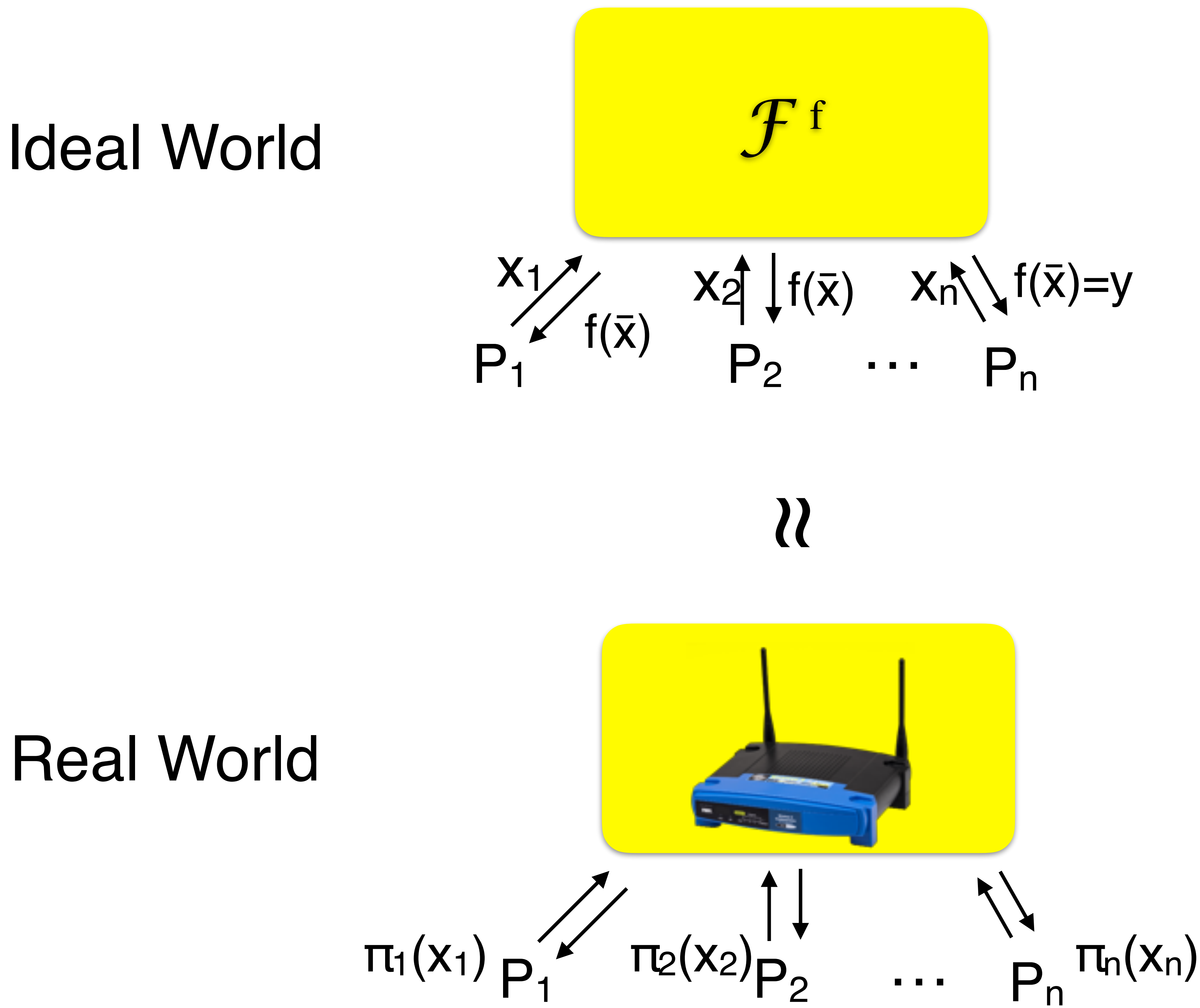
Ideal World



Real World

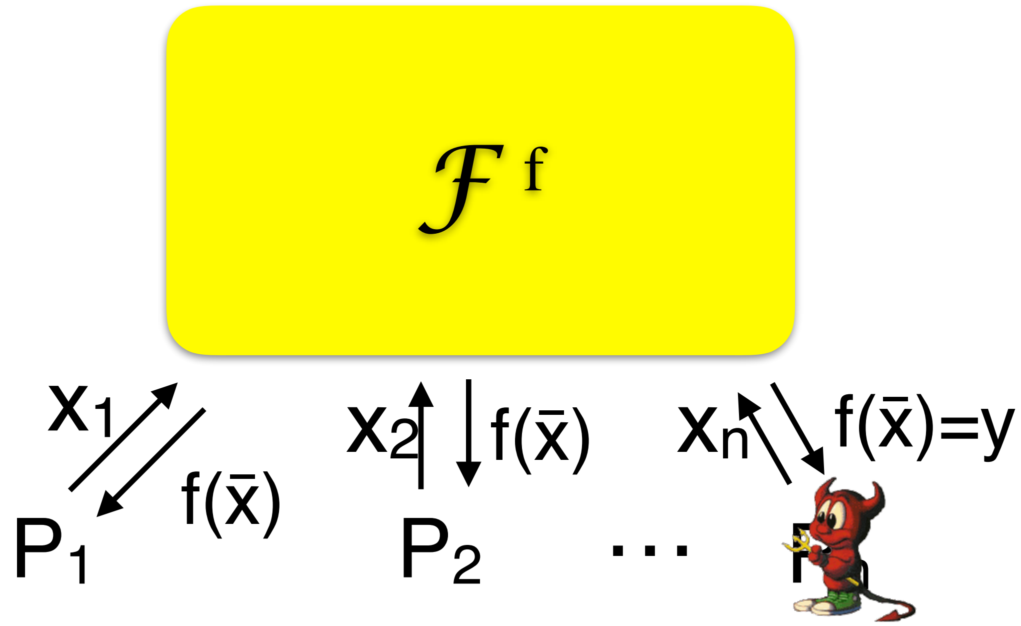


Secure Function Evaluation (SFE)



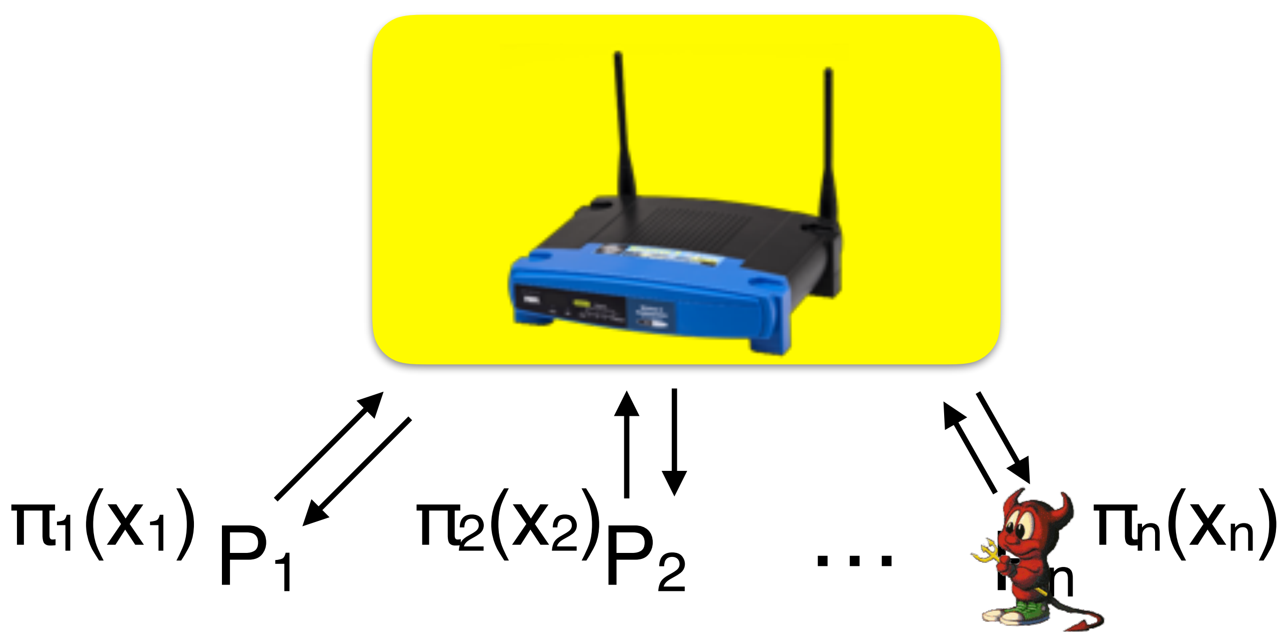
Secure Function Evaluation (SFE)

Ideal World

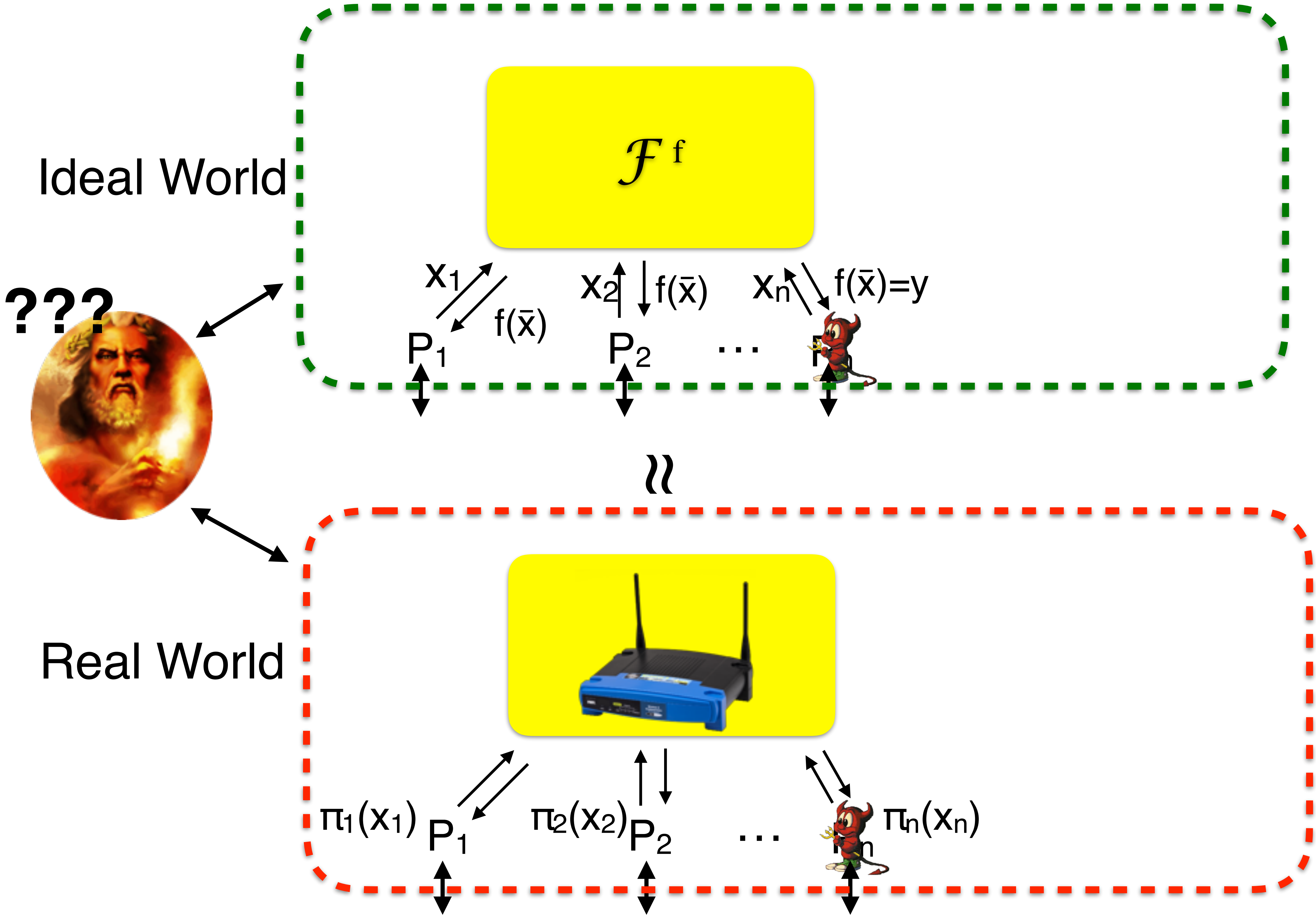


$\approx$

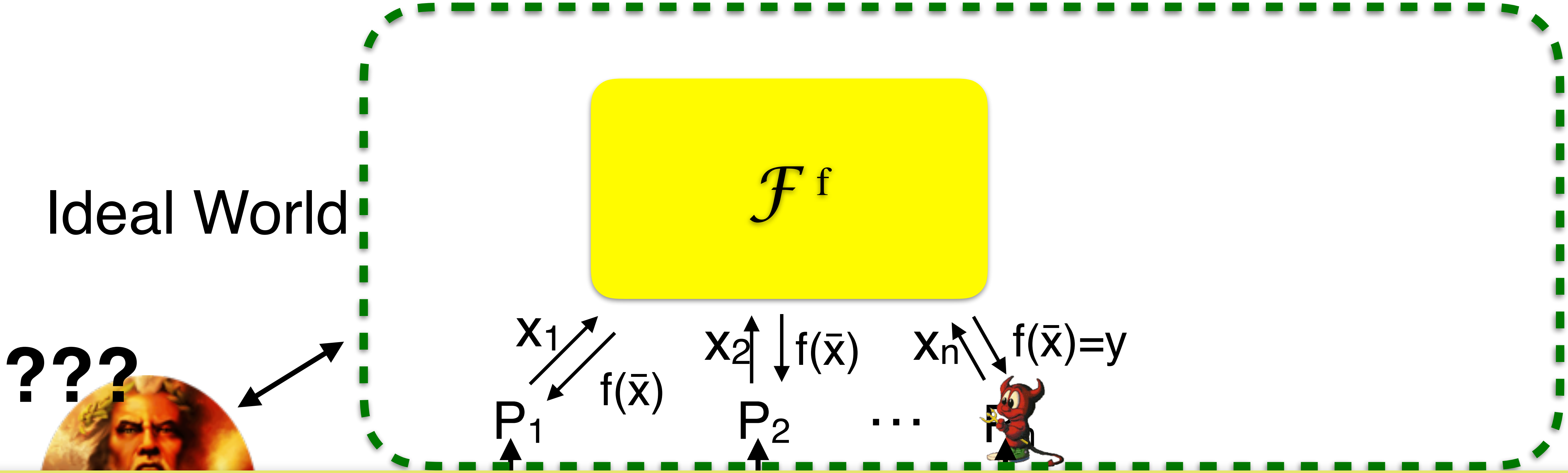
Real World



Secure Function Evaluation (SFE)

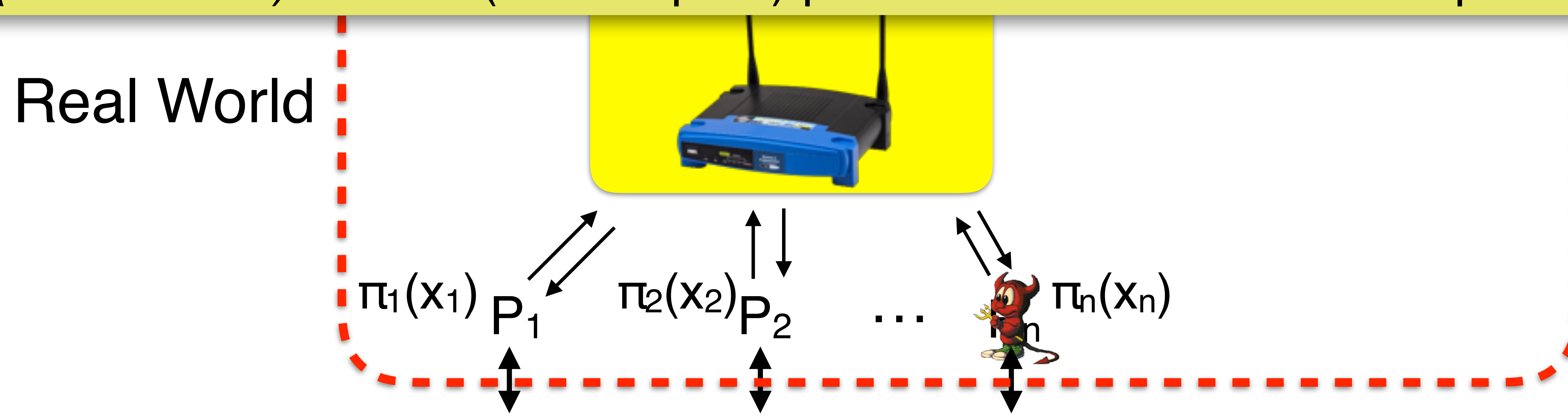


Secure Function Evaluation (SFE)



Protocol π is secure if *for every adversary*:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties learn their correct outputs

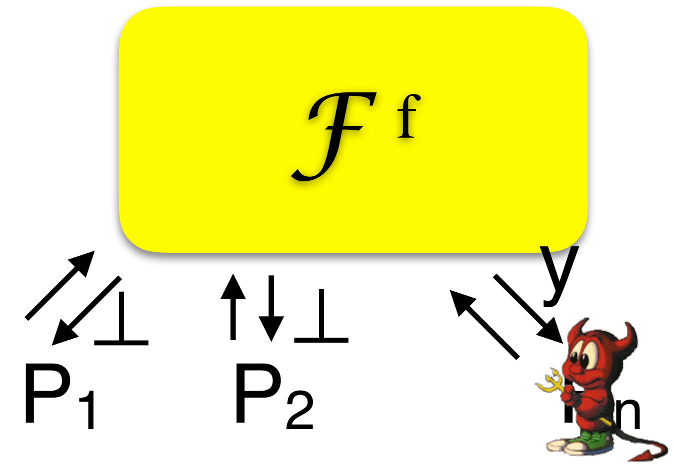


Fair SFE

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

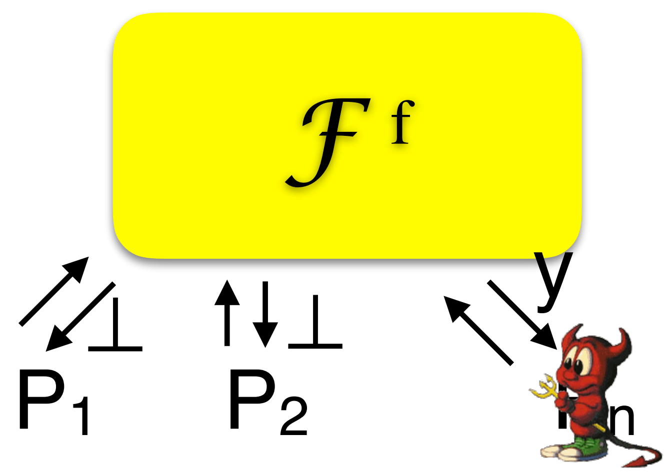
Fair SFE

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output



Fair SFE

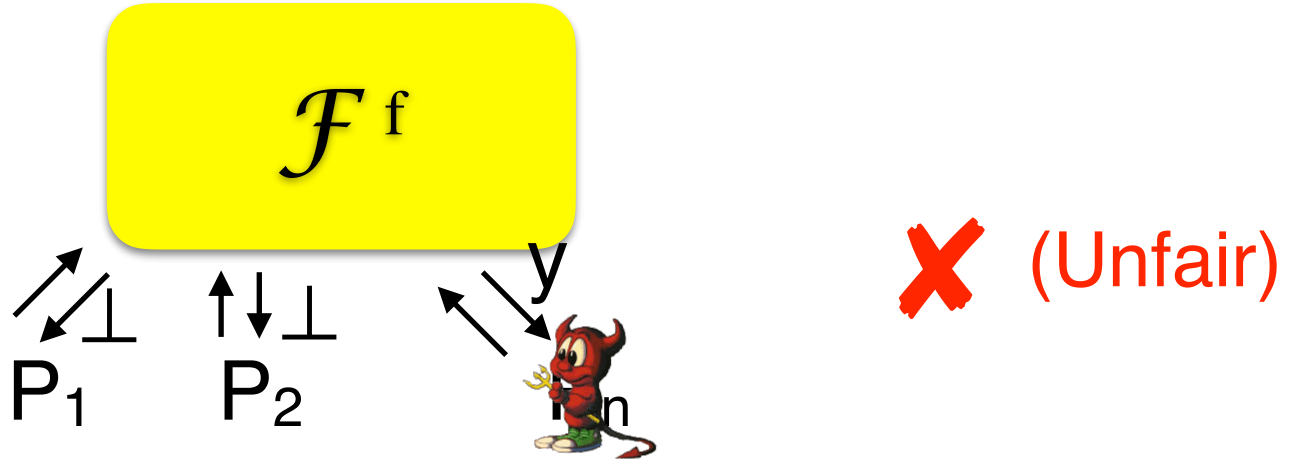
In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output



✘ (Unfair)

Fair SFE

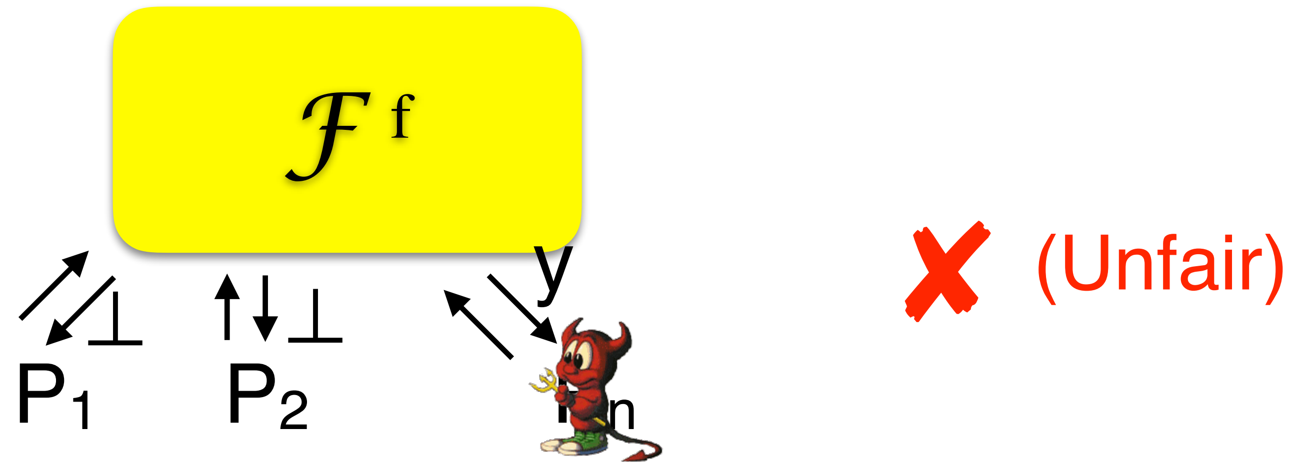
In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output



Fair SFE is impossible against corrupted majorities [Cleve86]

Fair SFE

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

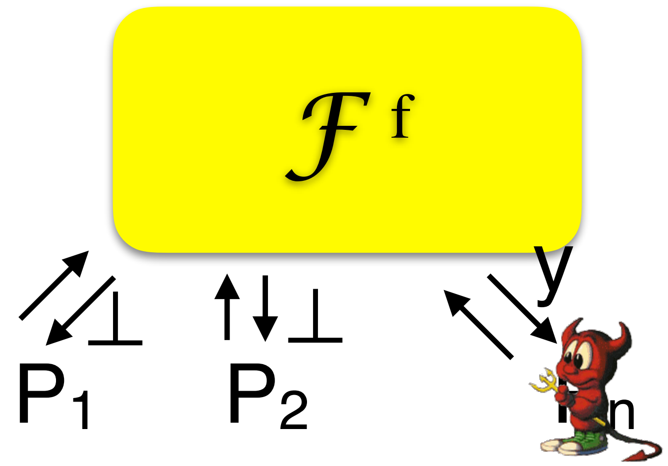


Fair SFE is impossible against corrupted majorities [Cleve86]

Security against corrupted majorities **=** **Security with abort**

Fair SFE

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output



✘ (Unfair)

Fair SFE is impossible against corrupted majorities [Cleve86]

Security against corrupted majorities **=** **Security with abort**

Discounted security

SFE with Fair(ness) Compensation

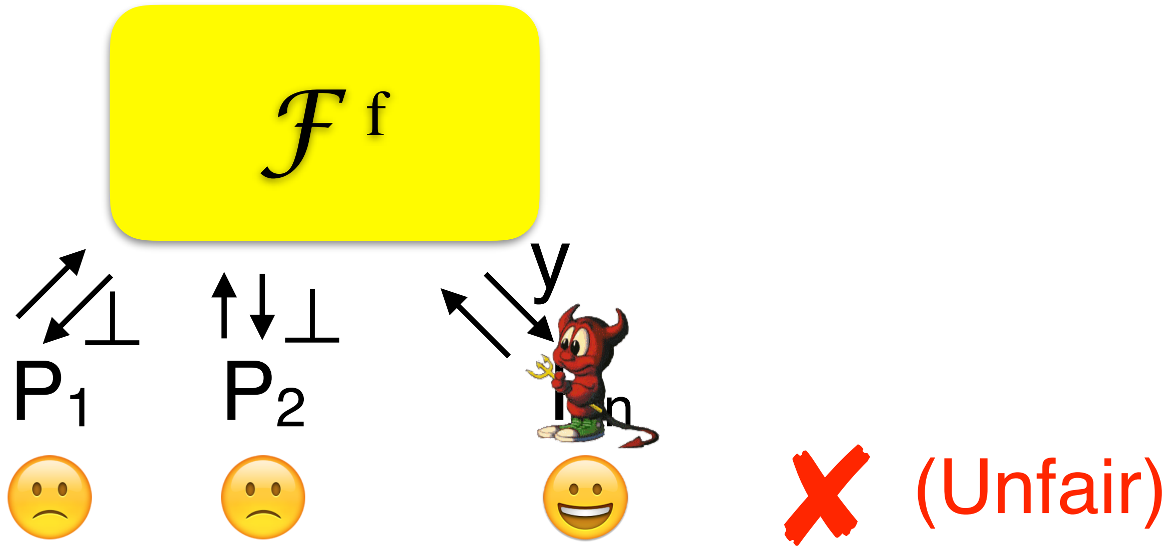
Idea [AndrychowiczDziembowskiMalinowskiMazurek14]:
We can leverage unfairness with $$$

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.

SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]:
We can leverage unfairness with $$$

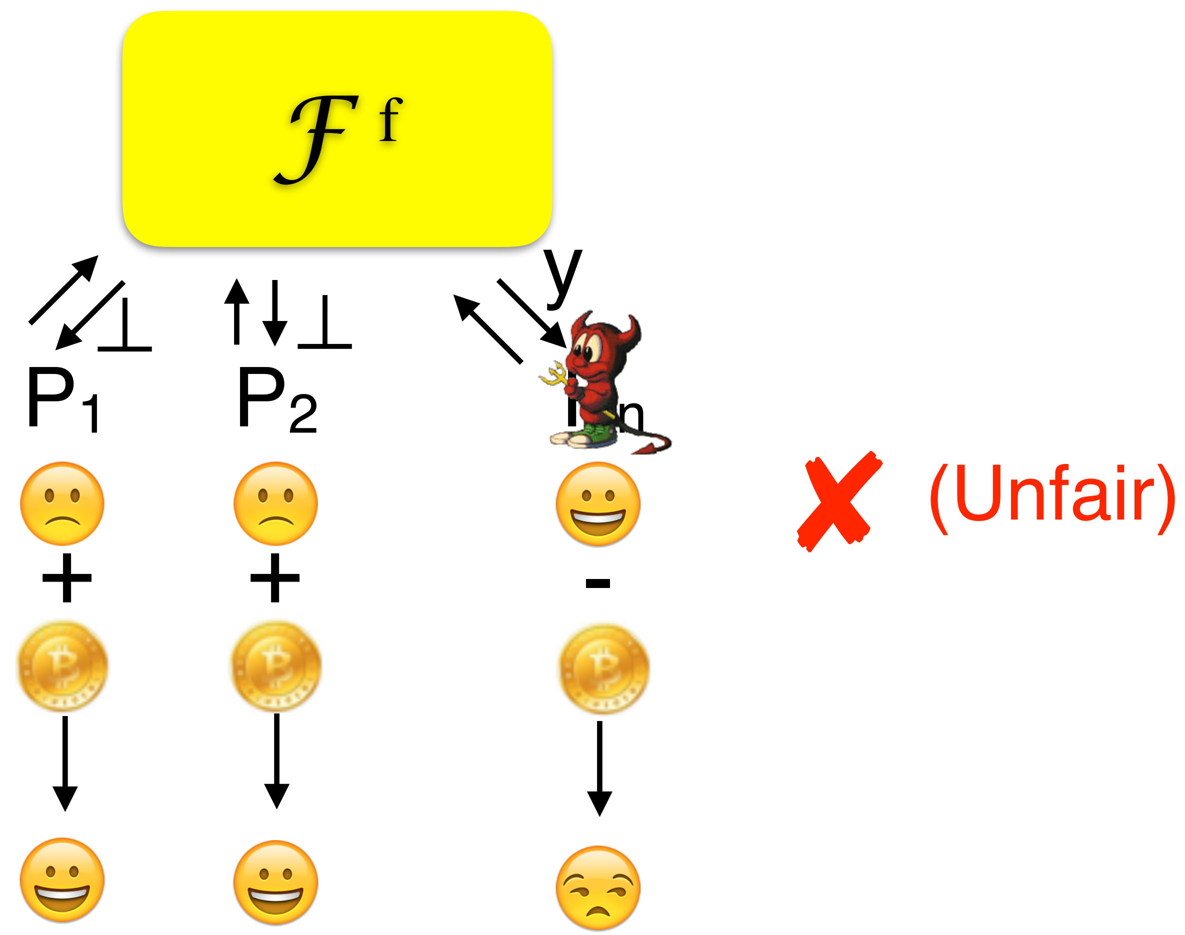
SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]:
We can leverage unfairness with $$$

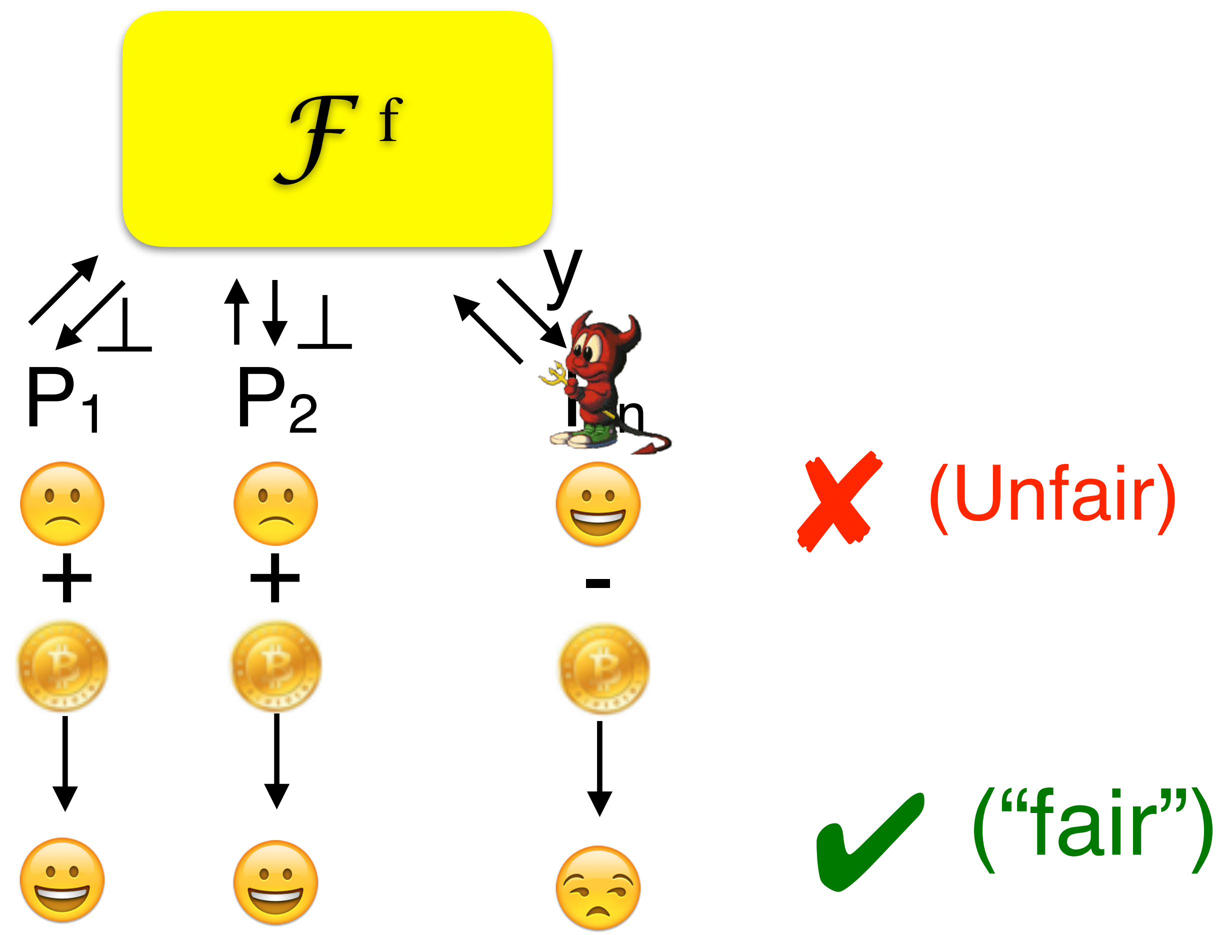
SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]:
We can leverage unfairness with $$$

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 1/2 : Authenticated Additive Secret Sharing

$x = x_1 \oplus \ldots \oplus x_n$, $(sk, vk) \leftarrow \text{KeyGen}$

P_1 … P_n

$[x]_1 = x_1, \text{Sig}_{sk}(x_1), vk$

$[x]_n = x_n, \text{Sig}_{sk}(x_n), vk$

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 1/2 : Authenticated Additive Secret Sharing

$x = x_1 \oplus \ldots \oplus x_n$, $(sk, vk) \leftarrow \text{KeyGen}$

P_1 … P_n

$[x]_1 = x_1, \text{Sig}_{sk}(x_1), vk$

$[x]_n = x_n, \text{Sig}_{sk}(x_n), vk$

- No *n-1* parties have info on x
- Together all *n* parties can recover x
- No party can lie about its share
  - Only x might be reconstructed!

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction τ

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction τ

time

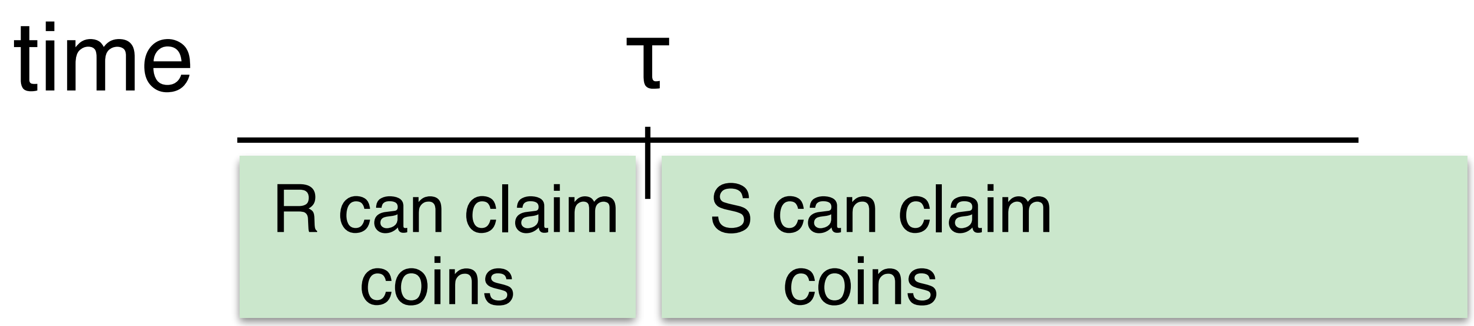
SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction τ



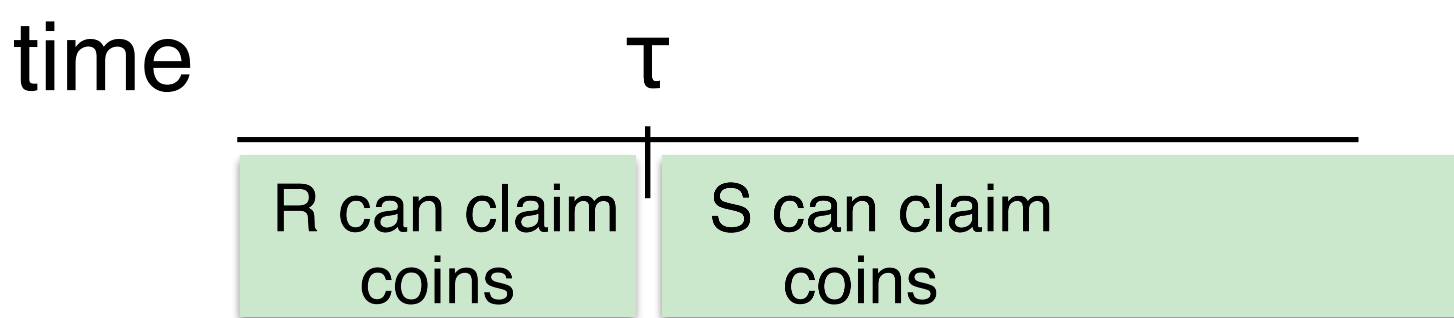
SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction τ



- A predicate (relation) $\mathcal{R}$(state,buffer,tx):
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

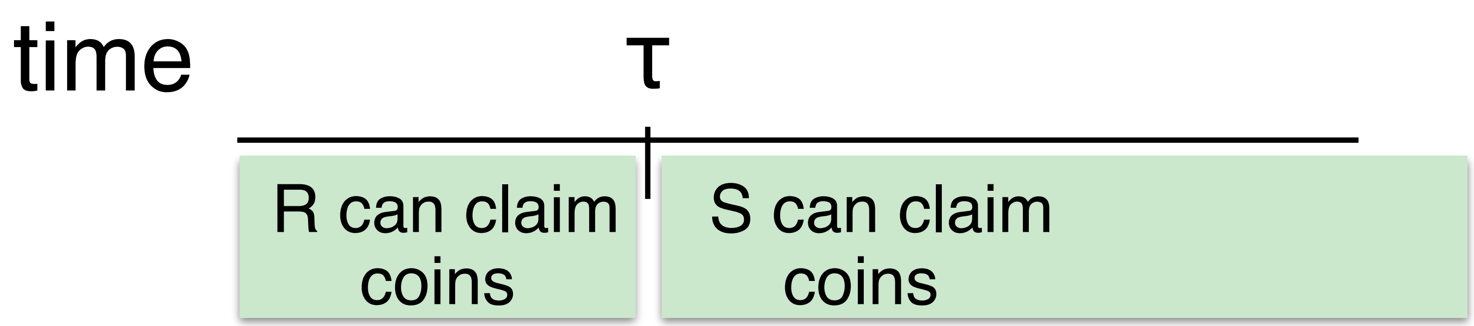
SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction τ



- A predicate (relation) $\mathcal{R}$(state,buffer,tx):
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

- Supported by Bitcoin scripting language
- Captured by *Validate(.)*

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

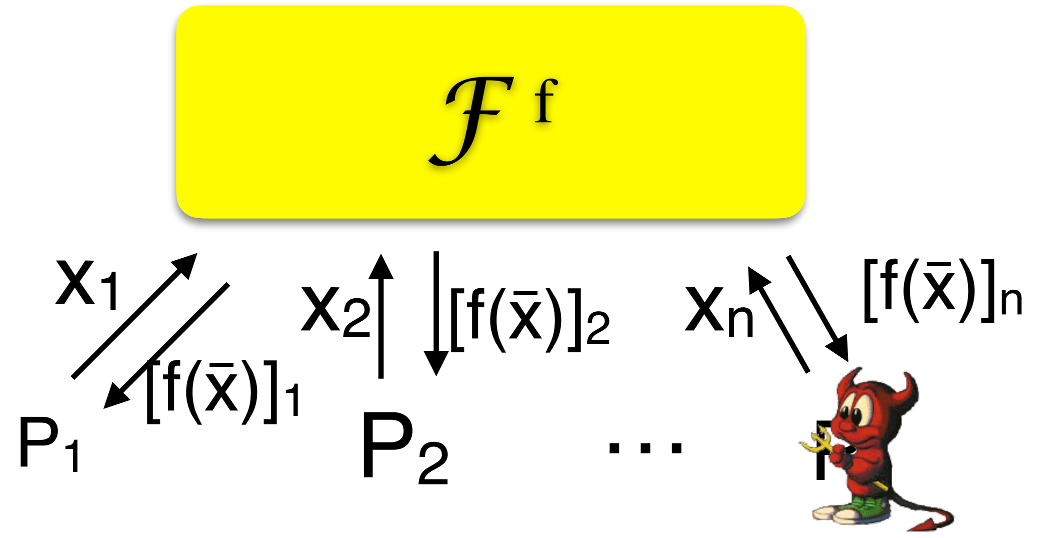
1. Run SFE with unfair abort to compute n-out-of-n authenticated sharing [y] of $y=f(x_1,\ldots,x_n)$
   - E.g., Every P_i receives share $[y]_i$ such that $y=[y]_1+\ldots+[y]_n$ and public signature on $[y]_i$

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

1. Run SFE with unfair abort to compute n-out-of-n authenticated sharing [y] of $y=f(x_1,\ldots,x_n)$
   - E.g., Every P_i receives share $[y]_i$ such that $y=[y]_1+\ldots+[y]_n$ and public signature on $[y]_i$

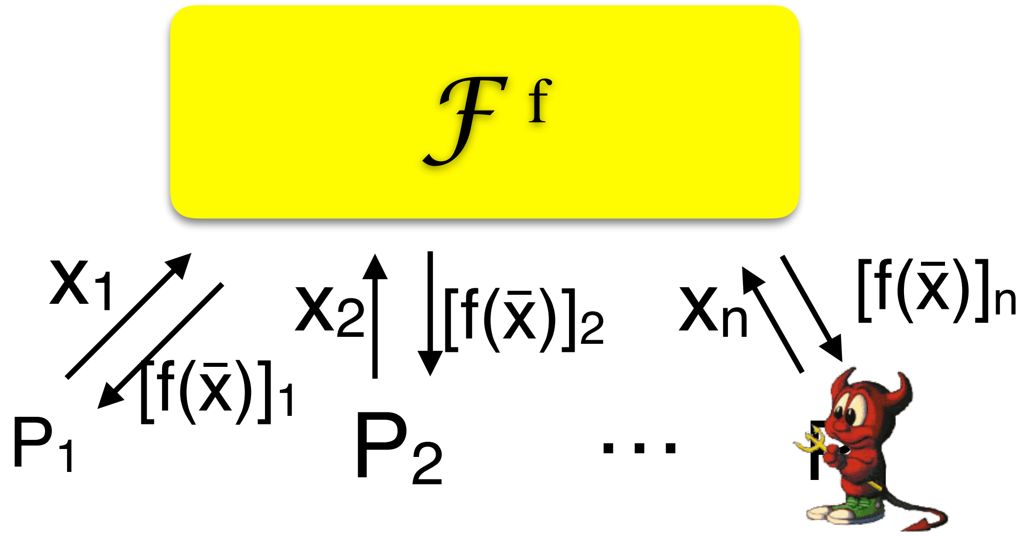


SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

1. Run SFE with unfair abort to compute n-out-of-n authenticated sharing [y] of $y=f(x_1,\ldots,x_n)$
   - E.g., Every P_i receives share $[y]_i$ such that $y=[y]_1+\ldots+[y]_n$ and public signature on $[y]_i$



…………………… Abort at this point is fair ……………………

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

2. Use the following reconstruction idea:

   2.1. Every P_i transfers 1 bitcoin to every P_j with the restriction:

   - P_j can claim (spend) this coin in round ρ_{ij} if it submits to the ledger his valid share (and signature) by round ρ_{ij}
   - if P_j has not claimed this coin by the end of round ρ_{ij}, then the coin is "refunded" to P_i (i.e., after round ρ_{ij}, P_i can spend this coin himself).

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

2. Use the following reconstruction idea:

   2.1. Every P_i transfers 1 bitcoin to every P_j with the restriction:

   - P_j can claim (spend) this coin in round ρ_{ij} if it submits to the ledger his valid share (and signature) by round ρ_{ij}
   - if P_j has not claimed this coin by the end of round ρ_{ij}, then the coin is "refunded" to P_i (i.e., after round ρ_{ij}, P_i can spend this coin himself).

   2.2. Proceed in rounds in which the parties claim the coins from other parties by announcing their shares (and signatures)

SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

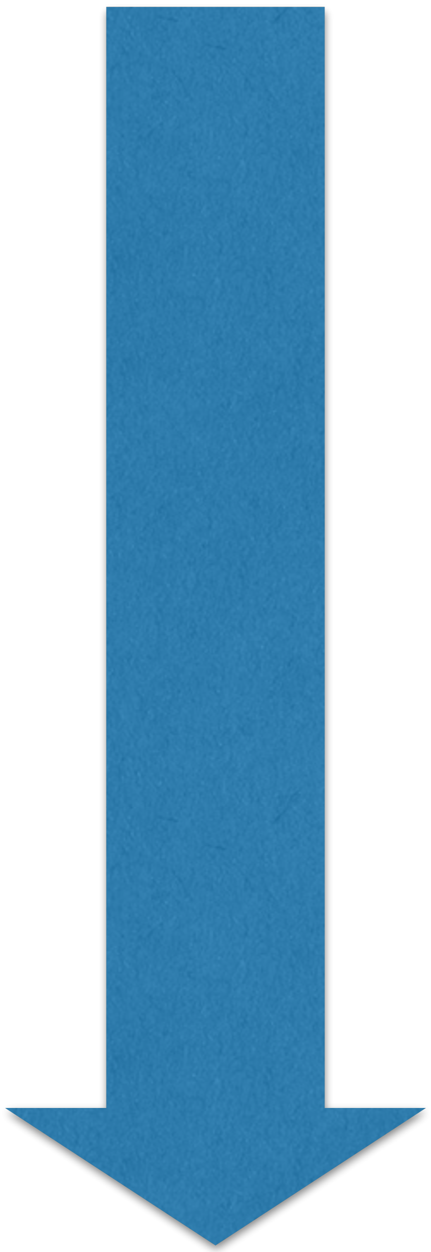
Security (SFE with fair compensation): Follow the money …

- If the adversary announces all his shares then every party:
  - Sends n coins in phase two (one to each party)
  - Claims back n coins in phase three (one from each party)
- If a corrupted party P_j does not announce his share then every party
  - Sends n coins in phase two (one to each party)
  - Claims back
    - n coins in phase three for announcing his shares
    - the coin that it had sent to P_j

Rethinking SFE w Fair(ness) Compensation

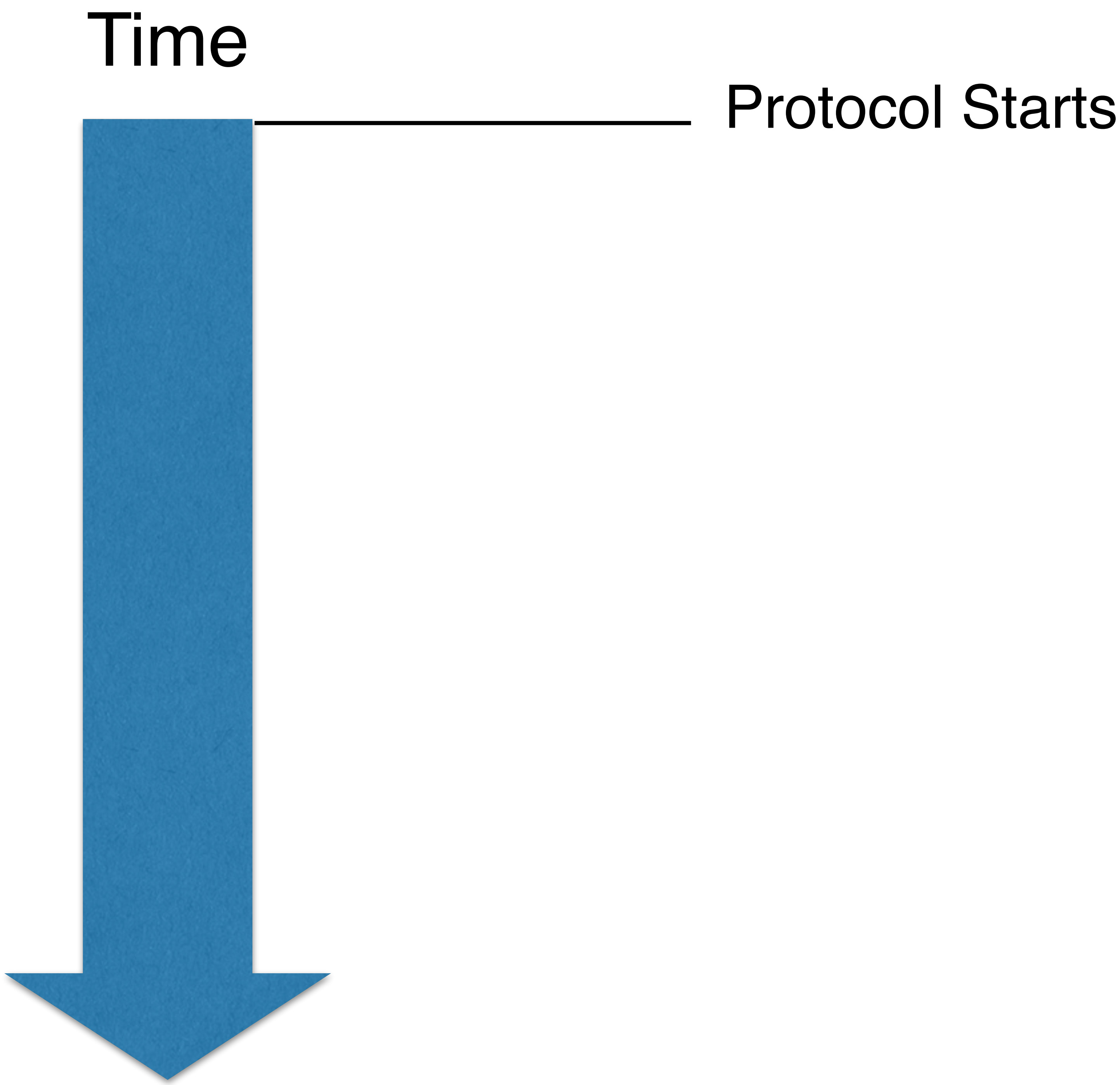
[BentovKumaresan14,15]

Time



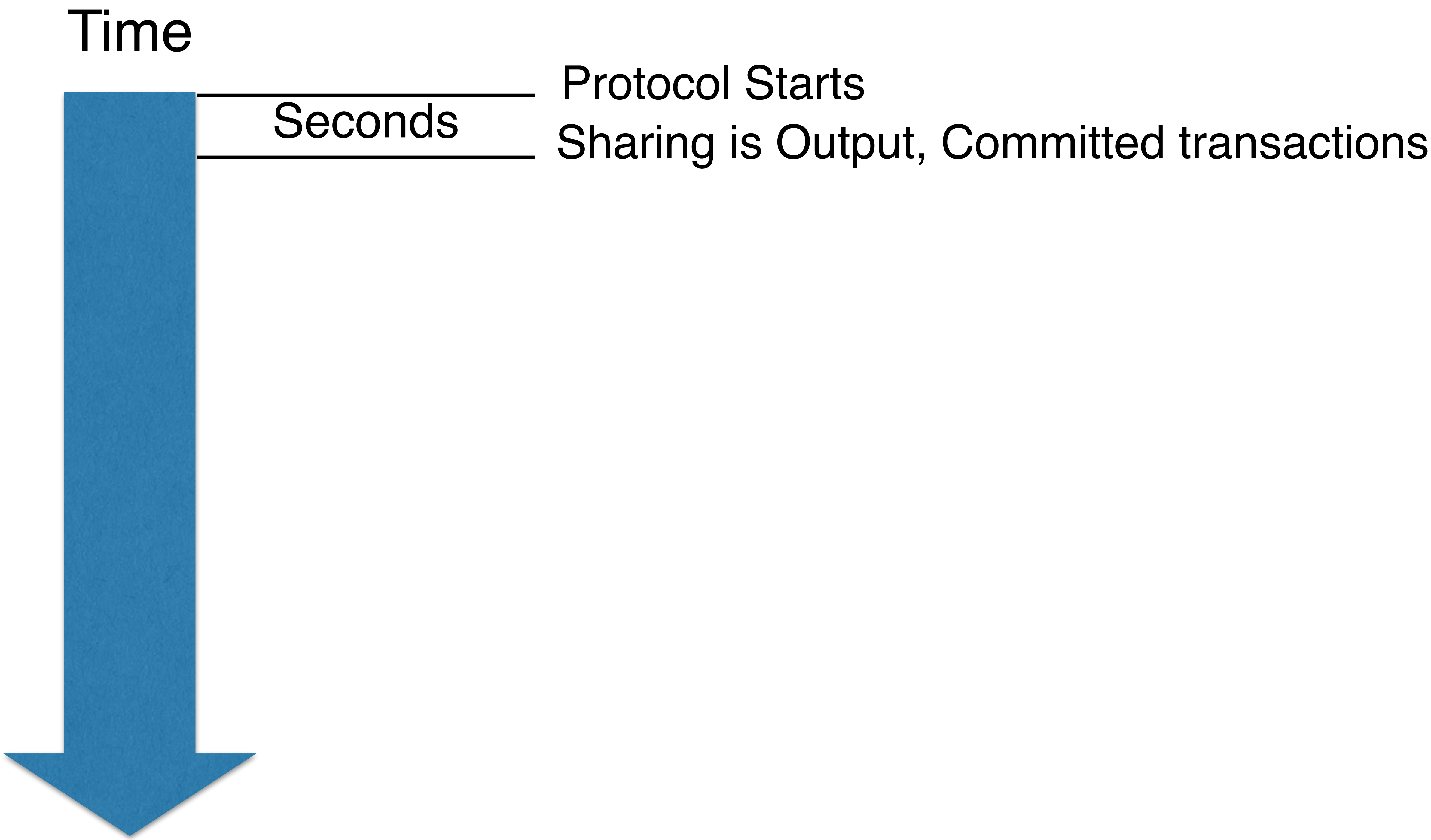
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]



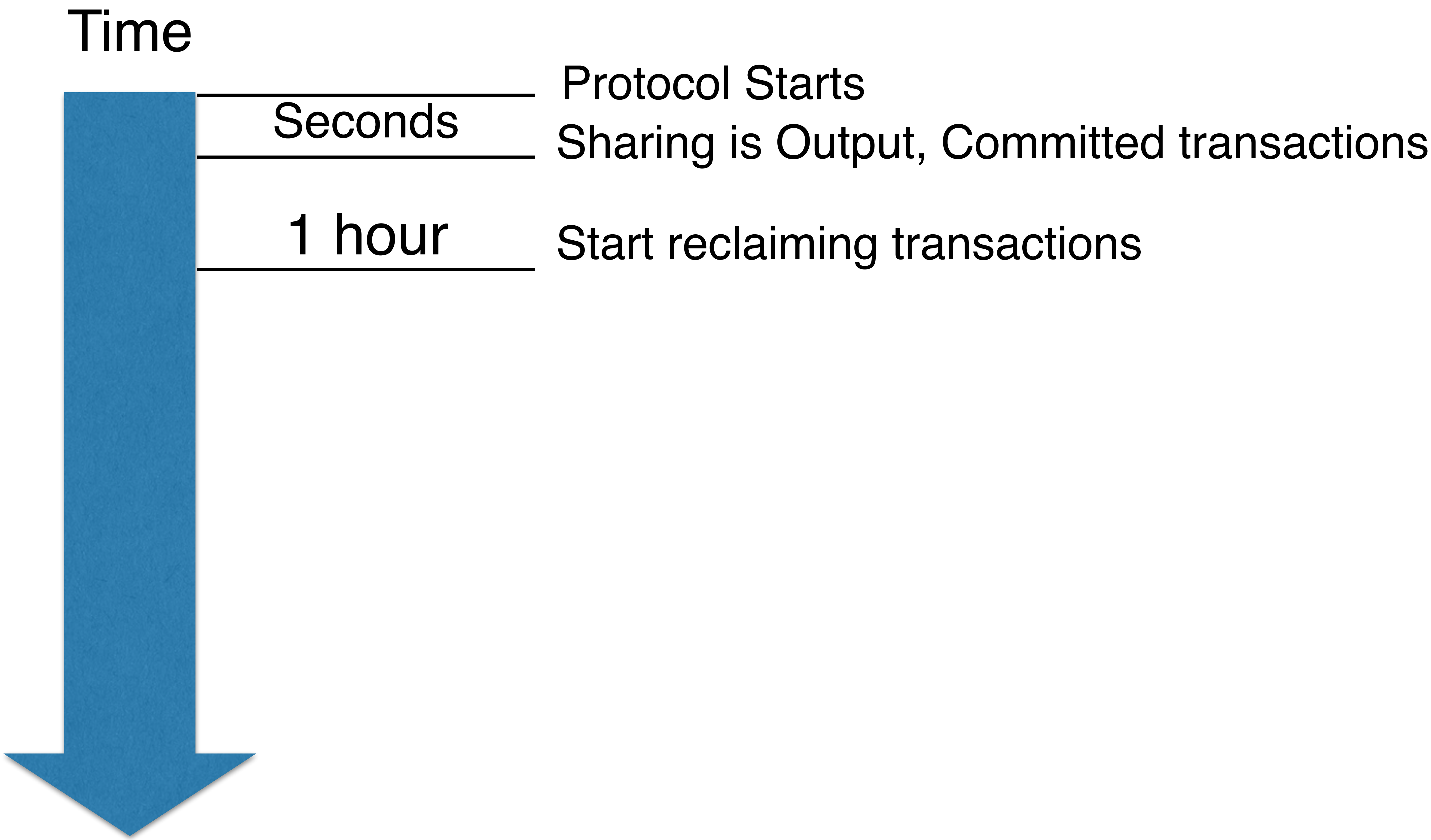
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]



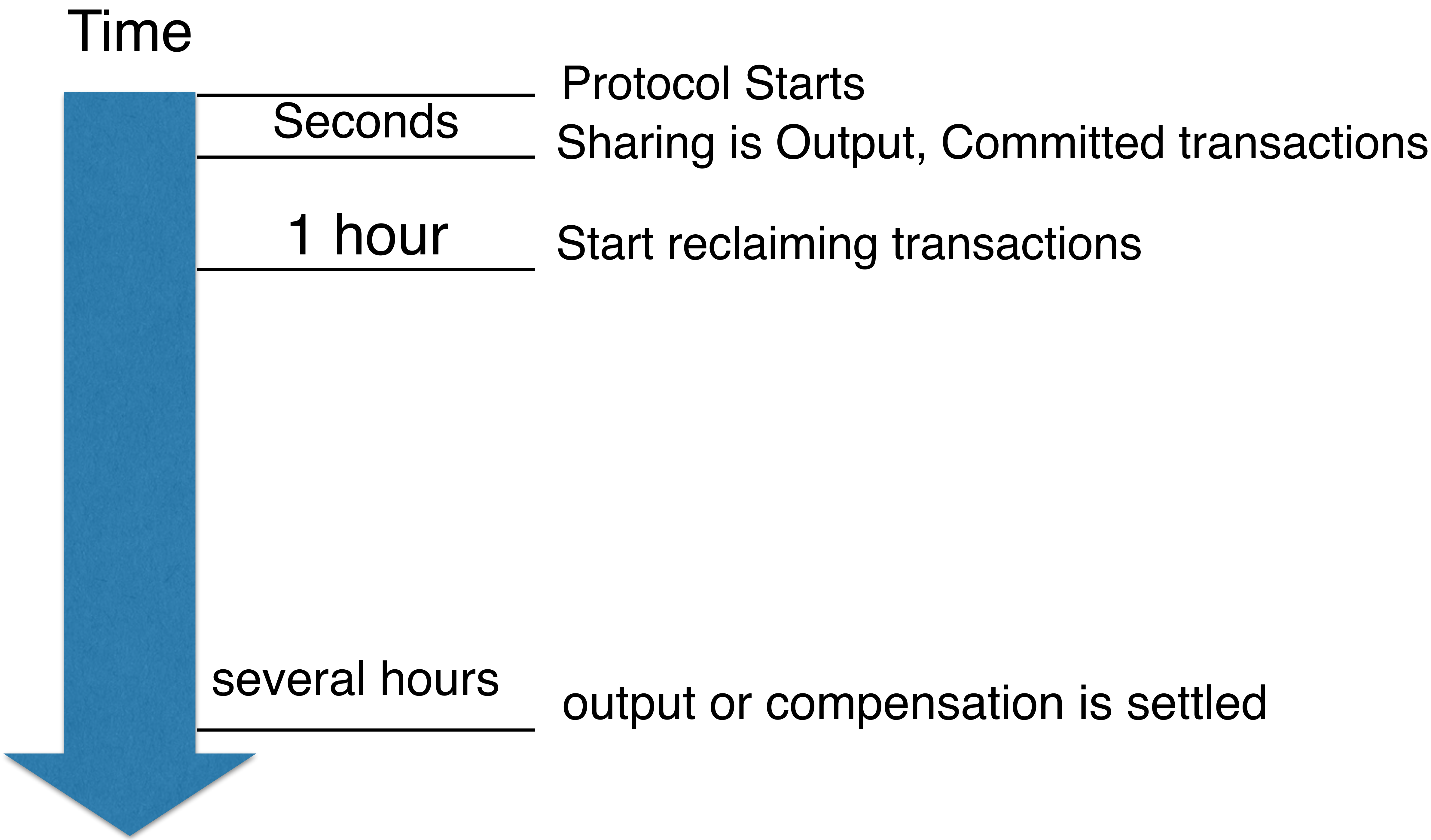
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]



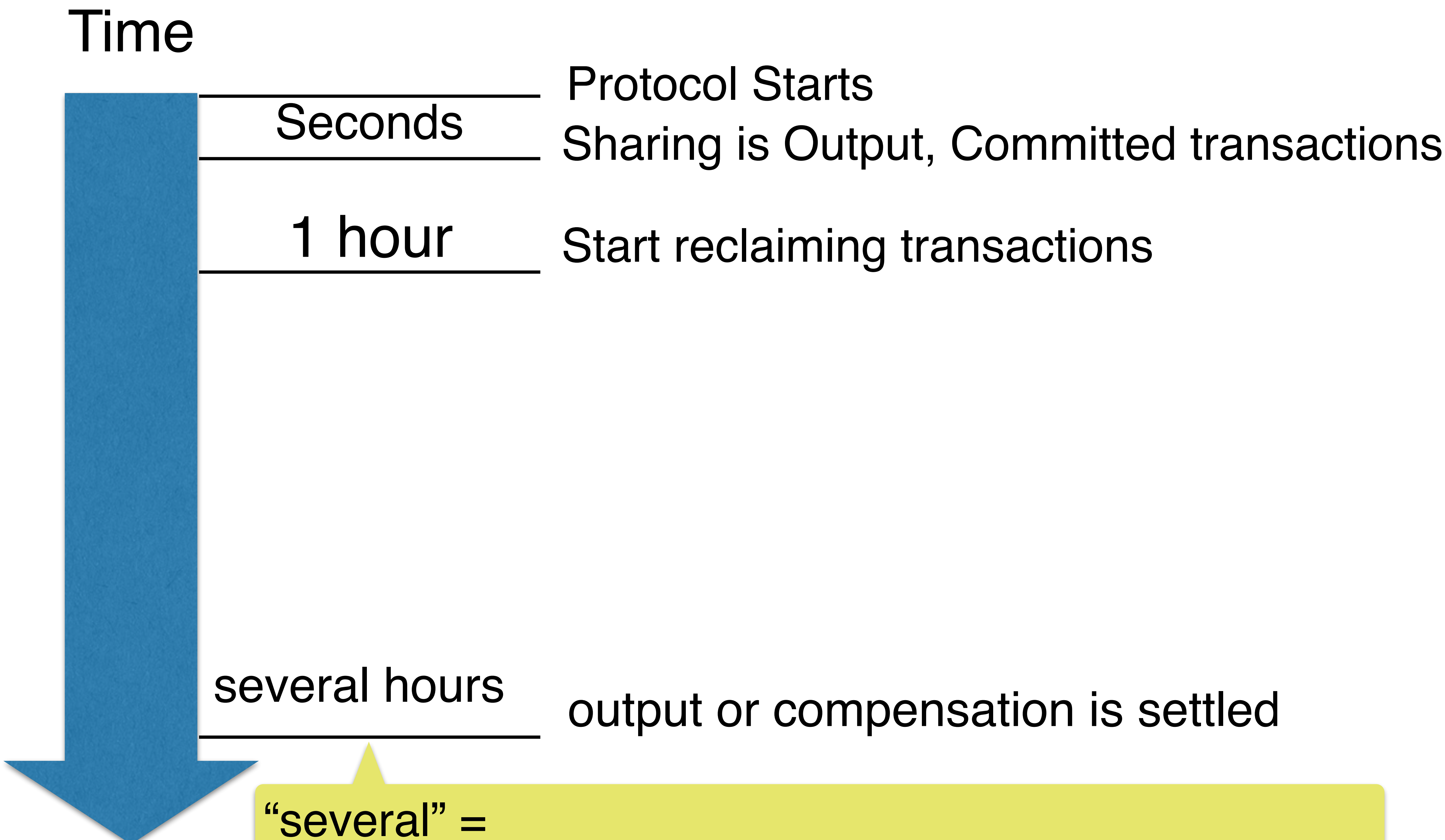
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]



Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]

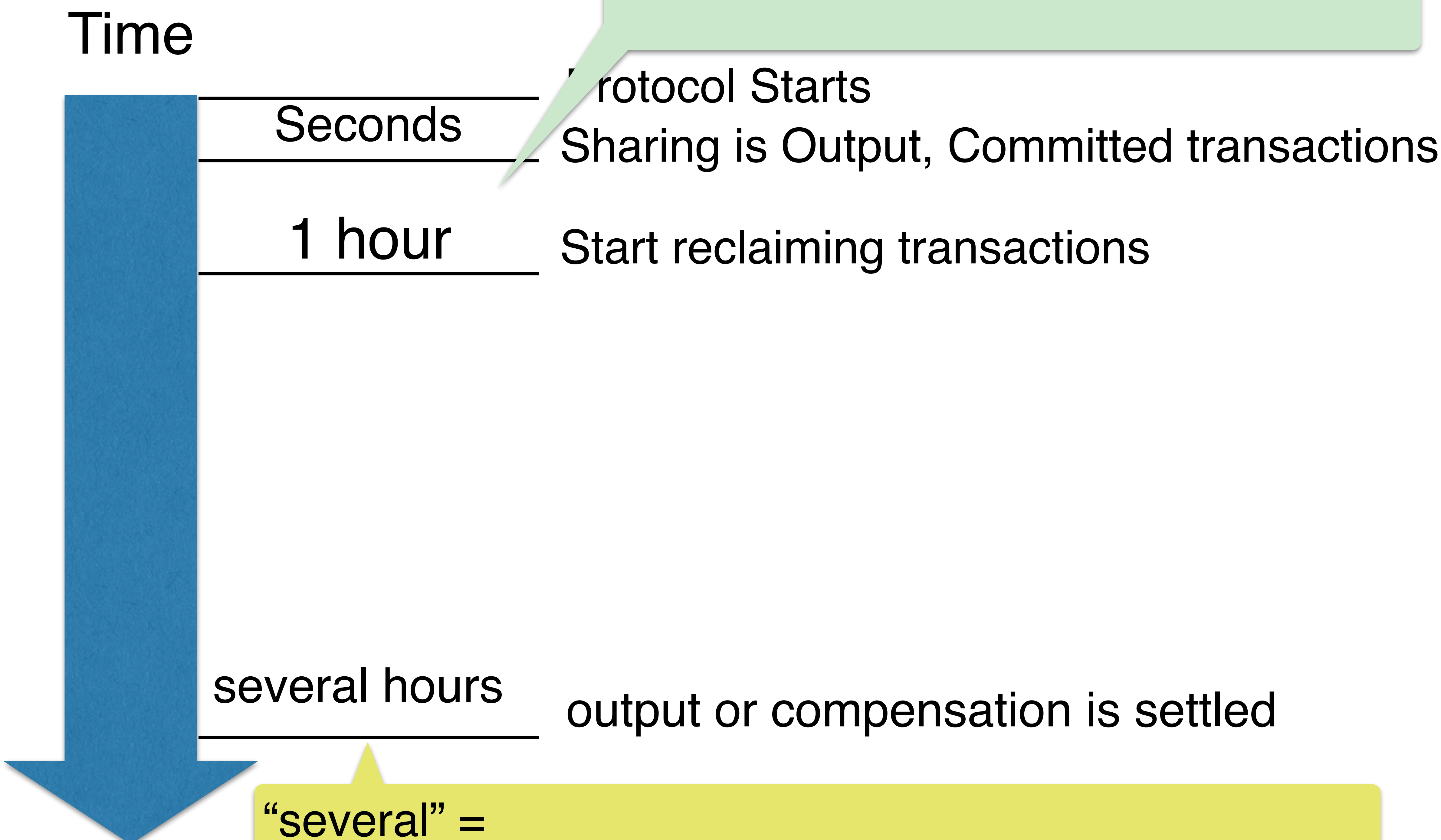


"several" =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?



"several" =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Time

Protocol Starts

Seconds

Sharing is Output, Committed transactions

1 hour

Start reclaiming transactions

This can be confirmed here …

several hours

output or compensation is settled

“several” =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Time

Seconds — Protocol Starts

Sharing is Output, Committed transactions

1 hour — Start reclaiming transactions

This can be confirmed here …

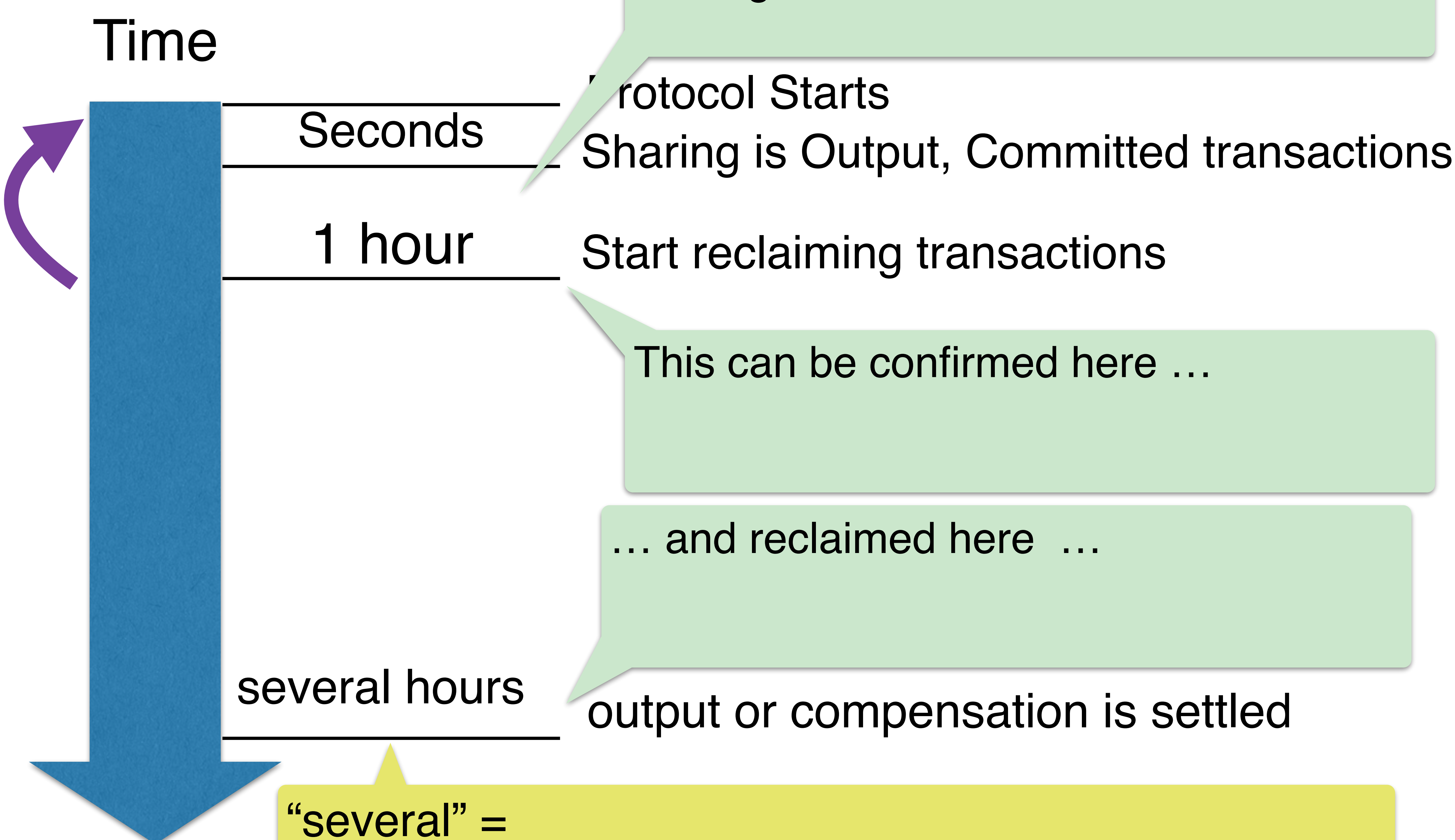
… and reclaimed here …

several hours — output or compensation is settled

“several” =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

Rethinking SFE w Fair(ness) Compensation



"several" =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Time

O(n) times = O(n) hours till output

Seconds — Protocol Starts

Sharing is Output, Committed transactions

1 hour — Start reclaiming transactions

This can be confirmed here …

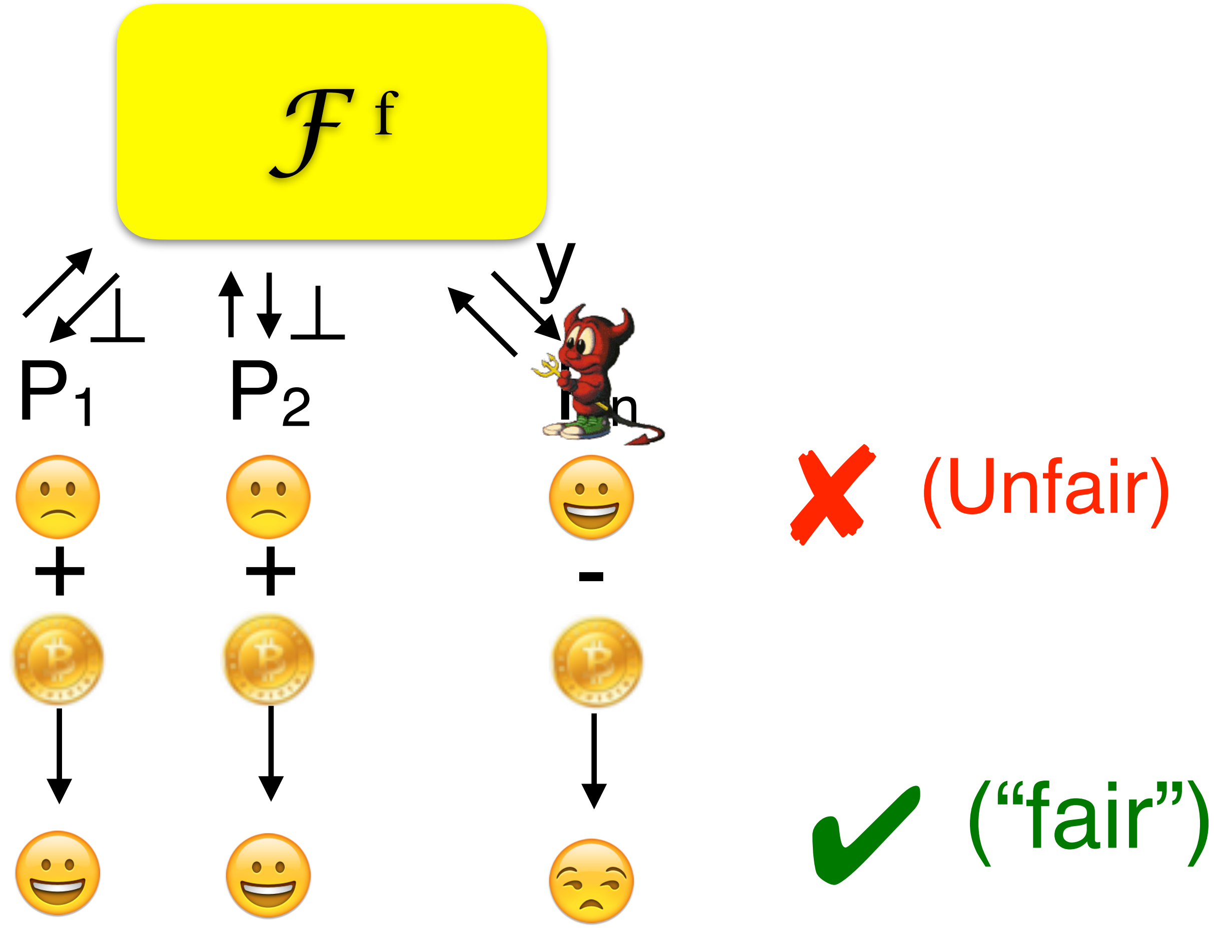
… and reclaimed here …

several hours — output or compensation is settled

"several" =
- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant

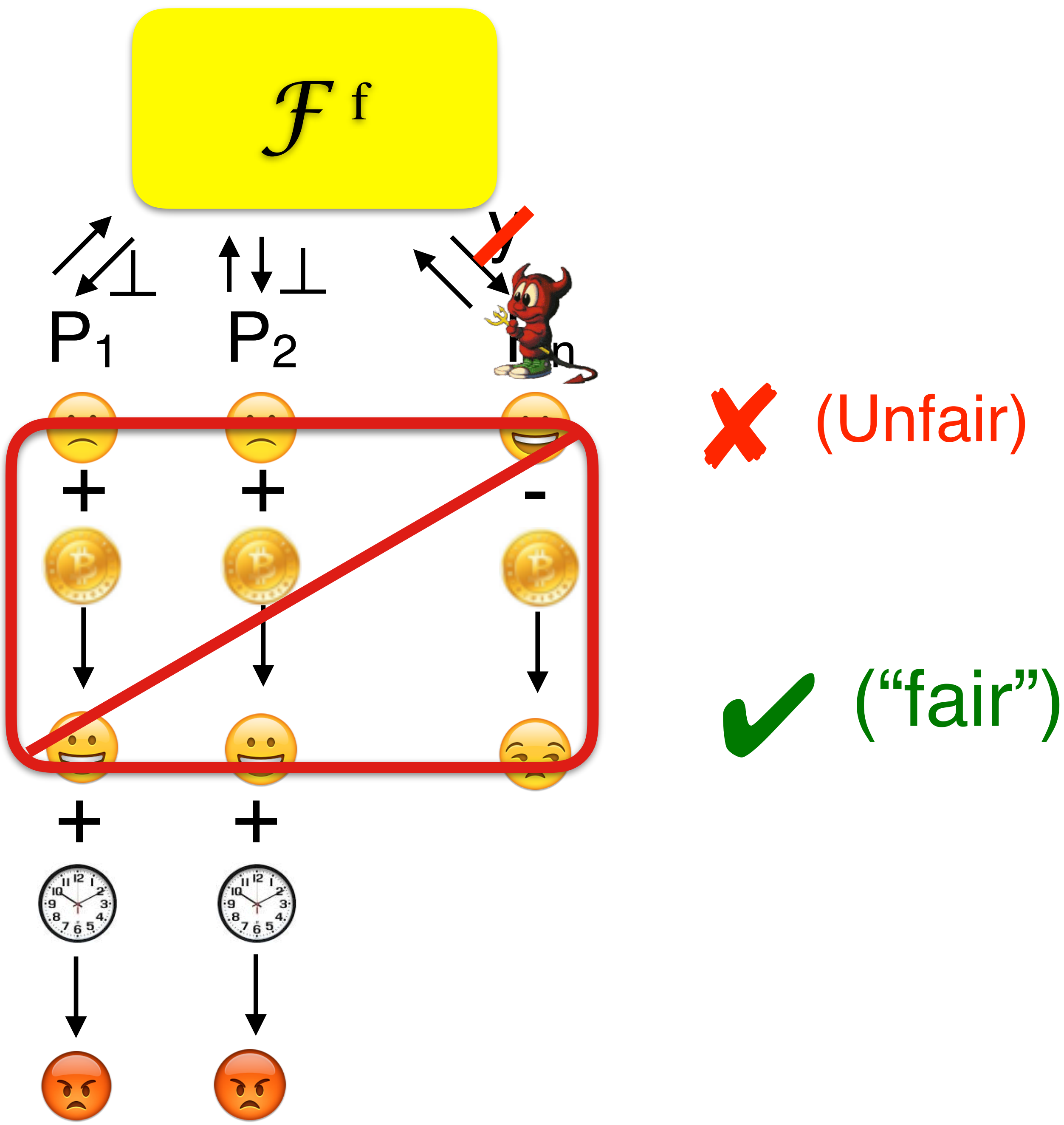
Rethinking SFE w Fair(ness) Compensation

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



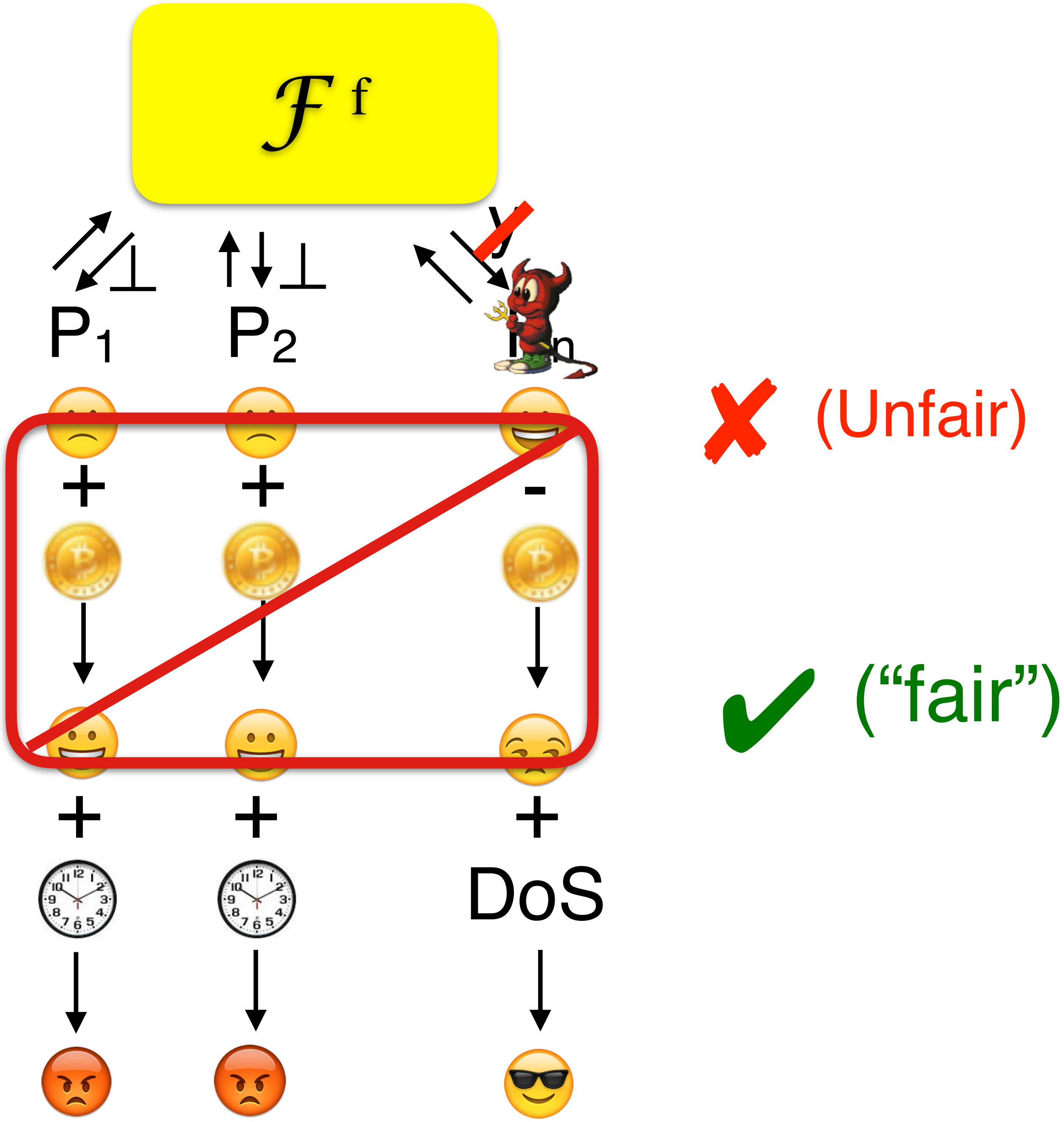
Rethinking SFE w Fair(ness) Compensation

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



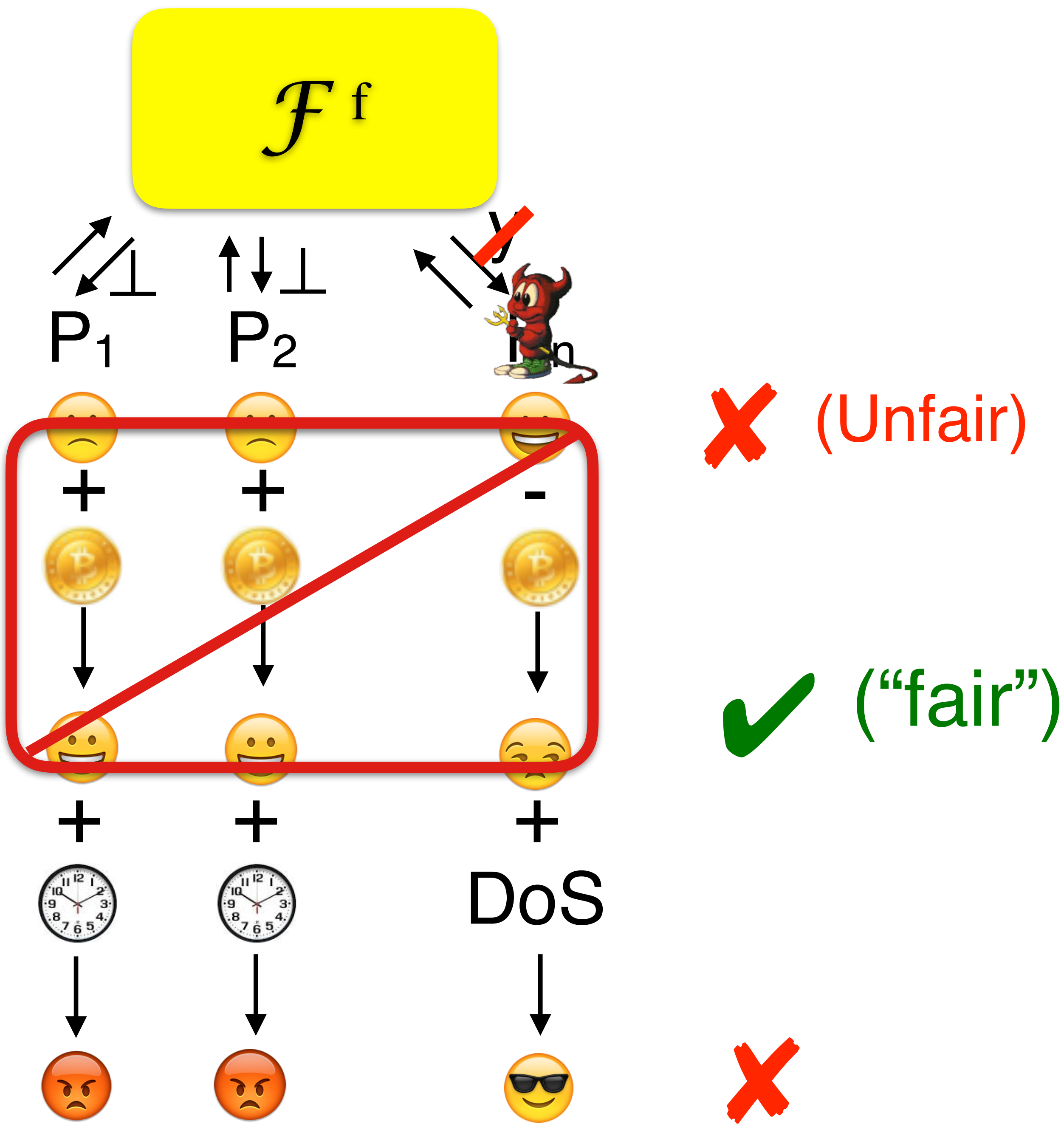
Rethinking SFE w Fair(ness) Compensation

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



Rethinking SFE w Fair(ness) Compensation

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.



SFE with Robust(ness) Compensation

SFE with Robust(ness) Compensation

Fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

SFE with Robust(ness) Compensation

Fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated

SFE with Robust(ness) Compensation

Fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

robust

SFE with ~~fair~~ compensation: ~~If the adversary learns any information beyond (what is derived by) its inputs then~~ every honest party should learn the output **or** get compensated (fast …)

SFE with Robust(ness) Compensation

Fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

robust

SFE with ~~fair~~ compensation: ~~If the adversary learns any information beyond (what is derived by) its inputs then~~ every honest party should learn the output **or** get compensated (fast …)

How can we get robustness?

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_{-}, τ_{+})

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)

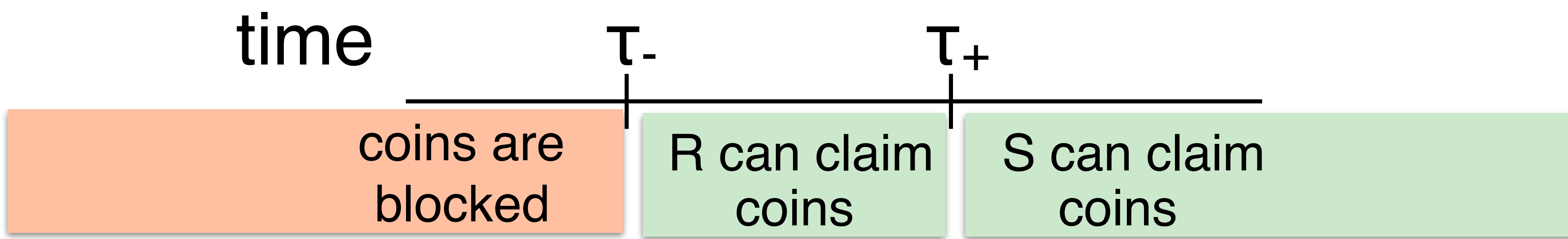
time

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)

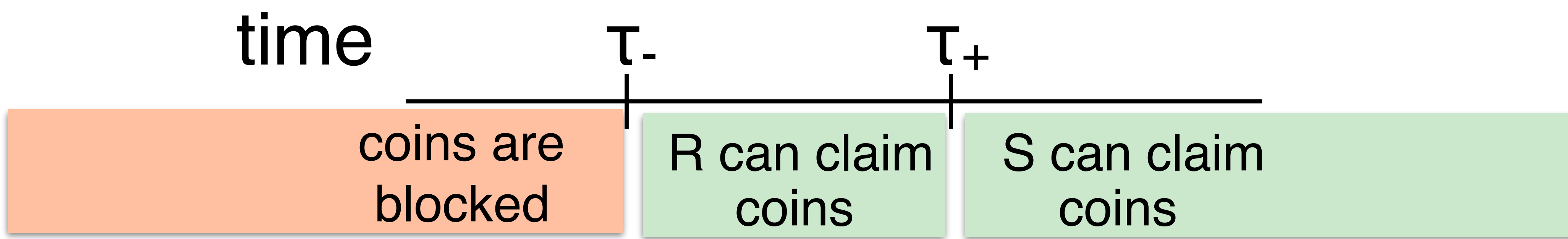


SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)



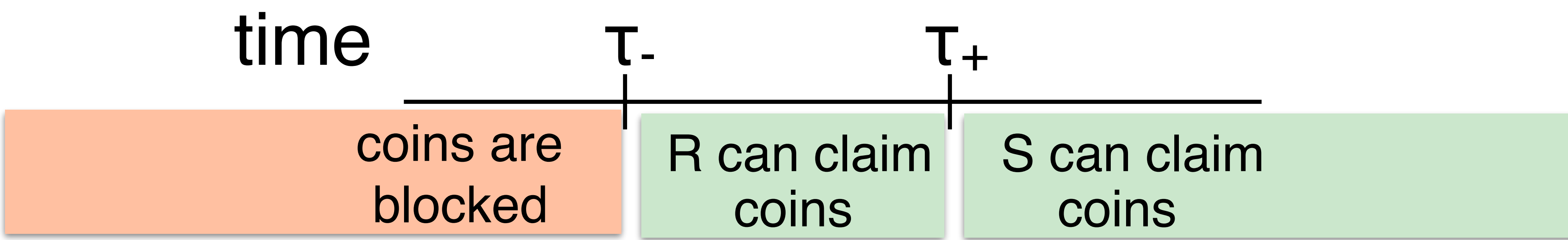
- Link: A reference ref such that only a transaction with the same reference can spend the q coins

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)



- Link: A reference ref such that only a transaction with the same reference can spend the q coins

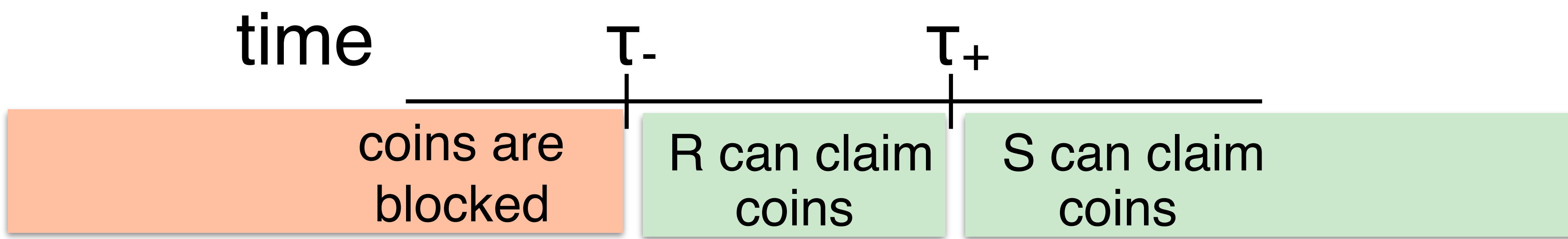
- A predicate (relation) $\mathcal{R}$(state,buffer,tx):
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)



- Link: A reference ref such that only a transaction with the same reference can spend the q coins

- A predicate (relation) $\mathcal{R}$(state,buffer,tx):
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

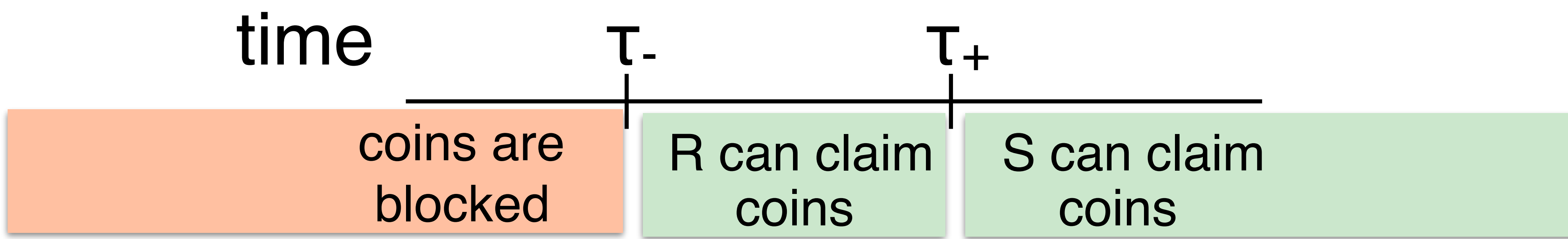
$$\mathbb{B}_{v,\mathtt{address}_i,\mathtt{address}_j,\Sigma,\mathtt{aux},\sigma_i,\tau}$$

SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction (τ_-, τ_+)



- Link: A reference ref such that only a transaction with the same reference can spend the q coins

- A predicate (relation) $\mathcal{R}$(state,buffer,tx):
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

(τ_-,τ_+), ref, $\mathcal{R}$

$$\mathbb{B}_{v,\mathtt{address}_i,\mathtt{address}_j,\Sigma,\mathtt{aux},\sigma_i,\tau}$$

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

P_1 x_1		
P_2 x_2		
$\vdots$		
P_n x_n		

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

	P_1	P_2		P_n	
P_1 x_1	x_{11}	x_{12}	$\cdots$	x_{1n}	$x_1 = \bigoplus_{j=1}^{n} x_{1j}$
P_2 x_2					
$\vdots$					
P_n x_n					

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

		P_1	P_2		P_n	
P_1	x_1	x_{11}	x_{12}	$\cdots$	x_{1n}	$x_1 = \bigoplus_{j=1}^{n} x_{1j}$
P_2	x_2	x_{21}	x_{22}	$\cdots$	x_{2n}	$x_2 = \bigoplus_{j=1}^{n} x_{2j}$
	$\vdots$		$\vdots$			$\vdots$
P_n	x_n	x_{n1}	x_{n2}	$\cdots$	x_{nn}	$x_n = \bigoplus_{j=1}^{n} x_{nj}$
		y_1	y_2	$\cdots$	y_n	

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

		P_1	P_2		P_n	
P_1	x_1	x_{11}	x_{12}	$\cdots$	x_{1n}	$x_1 = \bigoplus_{j=1}^{n} x_{1j}$
P_2	x_2	x_{21}	x_{22}	$\cdots$	x_{2n}	$x_2 = \bigoplus_{j=1}^{n} x_{2j}$
	$\vdots$		$\vdots$			$\vdots$
P_n	x_n	x_{n1}	x_{n2}	$\cdots$	x_{nn}	$x_n = \bigoplus_{j=1}^{n} x_{nj}$
		y_1	y_2	$\cdots$	y_n	$y = \bigoplus_{i=1}^{n} y_i$

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

Secure (private) against arbitrary many colluding parties

		P_1	P_2		P_n	
P_1	x_1	x_{11}	x_{12}	$\cdots$	x_{1n}	$x_1 = \bigoplus_{j=1}^{n} x_{1j}$
P_2	x_2	x_{21}	x_{22}	$\cdots$	x_{2n}	$x_2 = \bigoplus_{j=1}^{n} x_{2j}$
$\vdots$			$\vdots$			$\vdots$
P_n	x_n	x_{n1}	x_{n2}	$\cdots$	x_{nn}	$x_n = \bigoplus_{j=1}^{n} x_{nj}$
		y_1	y_2	$\cdots$	y_n	$y = \bigoplus_{i=1}^{n} y_i$

SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Assuming a public key infrastructure (commitments/encryption/ signatures) there exists a semi-honest SFE protocol π for **every function** which

- Uses only public communication
- Tolerates arbitrary many semi-honest parties
- Terminates in constant rounds

SFE with Robust Compen. : Construction

Tools 3/3 : The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

SFE with Robust Compen. : Construction

Tools 3/3 : The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every P_i commits to its input

Rounds 2 … $\rho_\pi + 1$:
Execute π round-by-round so that in each round every party proves (in ZK) that he follows π

SFE with Robust Compen. : Construction

Tools 3/3 : The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every P_i commits to its input

Rounds 2 … $\rho_\pi + 1$:
Execute π round-by-round so that in each round every party proves (in ZK) that he follows π

Security (with abort)

- **Privacy:** The parties see the following:
  - Setup
  - Commitments
  - Messages from π
- **Correctness:**
  - If ZKPs succeed then the parties are indeed following π
  - Else abort

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every P_i commits to its input

Rounds 2 … $\rho_\pi + 1$:
Execute π round-by-round so that in each round every party proves (in ZK) that he follows π

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW → **GMW':**

GMW	GMW'
Round 0: Setup generation (+ commitments to randomness)	**Round 0:** Setup generation (+ commitments to randomness)
	Round 1: Do nothing
Round 1: Every P_i commits to its input	**Round 2:** Every P_i commits to its input and broadcasts his view of the public setup.
Rounds 2 … $\rho_\pi + 1$: Execute π round-by-round so that in each round every party proves (in ZK) that he follows π	**Rounds 3 … $\rho_\pi + 2$:** Execute π round-by-round so that in each round every party proves (in NIZK) that he follows π

SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

GMW’:

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

GMW’: → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

Round 0:
Setup generation (+ commitments to randomness)

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

Round 0:
Setup generation (+ commitments to randomness)

Round 1: Every party P_i makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:

- P_j can spend coin in round r
- ref needs to have the protocol ID
- R is true if the transaction which spends the coin includes a valid r-round message for P_j

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

GMW':

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

SFE with Robust Compensation:

Round 0:
Setup generation (+ commitments to randomness)

Round 1: Every party P_i makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:

- P_j can spend coin in round r
- ref needs to have the protocol ID
- R is true if the transaction which spends the coin includes a valid r-round message for P_j

Rounds 3 … $\rho_\pi + 2$: Execute GMW(π) round-by-round so that in each round r every party spends all its round r referenced coins by a transaction which includes the round r message in GMW(π).

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that

`Validate(.)` executes the code of an extra party without inputs in GMW and rejects if abort.

Round 0:
Setup generation (+ commitments to randomness)

Round 1: Every party P_i makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:

- P_j can spend coin in round r
- ref needs to have the protocol ID
- R is true if the transaction which spends the coin includes a valid r-round message for P_j

Rounds 3 … $\rho_\pi + 2$: Execute GMW(π) round-by-round so that in each round r every party spends all its round r referenced coins by a transaction which includes the round r message in GMW(π).

SFE with Robust Compen. : Construction

Security with Robust Compensation.

- **Case 1:** The adversary correctly makes all the "committing" transactions in Round 1
    - If no party cheats then every party claims from each of the other parties as many coins as he deposited by simply executing his protocol.
    - If some party P_j cheats, then every party still claims all his coins as above + all the committed coins that P_j cannot spend as he did not execute his protocol.

SFE with Robust Compen. : Construction

Security with Robust Compensation.

- **Case 2:** Some corrupted party does not make (consistent) transactions in Round 1
  - e.g. aborts or commits to a different setup.

SFE with Robust Compen. : Construction

Security with Robust Compensation.

- **Case 2:** Some corrupted party does not make (consistent) transactions in Round 1
  - e.g. aborts or commits to a different setup.

    … seems to have similar issue as before …

SFE with Robust Compen. : Construction

Security with Robust Compensation.

- **Case 2:** Some corrupted party does not make (consistent) transactions in Round 1
  - e.g. aborts or commits to a different setup.

  … seems to have similar issue as before …

- Solution: The validation predicate can be changed as:
  - Separates the parties into “islands” of consistent setups (depending on their Round-1 transactions).
  - For each island $I \subseteq [n]$: Compute the function among parties in I (with all other parties’ input being 0)

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW': → **SFE with Robust Compensation**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

SFE with Robust Compensation

Round 0:
Setup generation (+ commitments to randomness)

Round 1: Every party P_i makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:

- P_j can spend coin in round r
- ref needs to have the protocol ID
- R is true if the transaction which spends the coin includes a valid r-round message for P_j

Rounds 2 … $\rho_\pi + 2$: Execute GMW(π) round-by-round so that in each round r every party spends all its round r referenced coins by a transaction which includes the round r message in GMW(π).

SFE with Robust Compen. : Construction

Idea: Use "GMW"-like compiler on the Ledger

GMW':

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every P_i commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_\pi + 2$:
Execute π round-by-round so that in each round every party proves (in NIZK) that the follows π

→

SFE with Robust Compensation

Round 0:
Setup generation (+ commitments to randomness)

Round 1: Every party P_i makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:

- P_j can spend coin in round r
- ref needs to have the protocol ID
- R is true if the transaction which spends the coin includes a valid r-round message for P_j

Rounds 2 … $\rho_\pi + 2$: Execute GMW(π) round-by-round so that in each round r every party spends all its round r referenced coins by a transaction which includes the round r message in GMW(π).

SFE with Robust Compen. : Construction

Security with Robust Compensation.

- **Case 2:** Some corrupted party does not make (consistent) transaction in Round 1
  - e.g. aborts or commits to a different setup.

… seems to have similar issue as before …

- Solution: The validation predicate can be changed as:
  - Separates the parties into "islands" of consistent setups (depending on their Round-1 transactions).
  - For each island I⊆[n]: Compute the function among parties in I (with all other parties' input being 0)

- All honest parties are on the same island
- Corrupted parties can choose to play with the honest parties or participate in a computation independent of honest inputs.

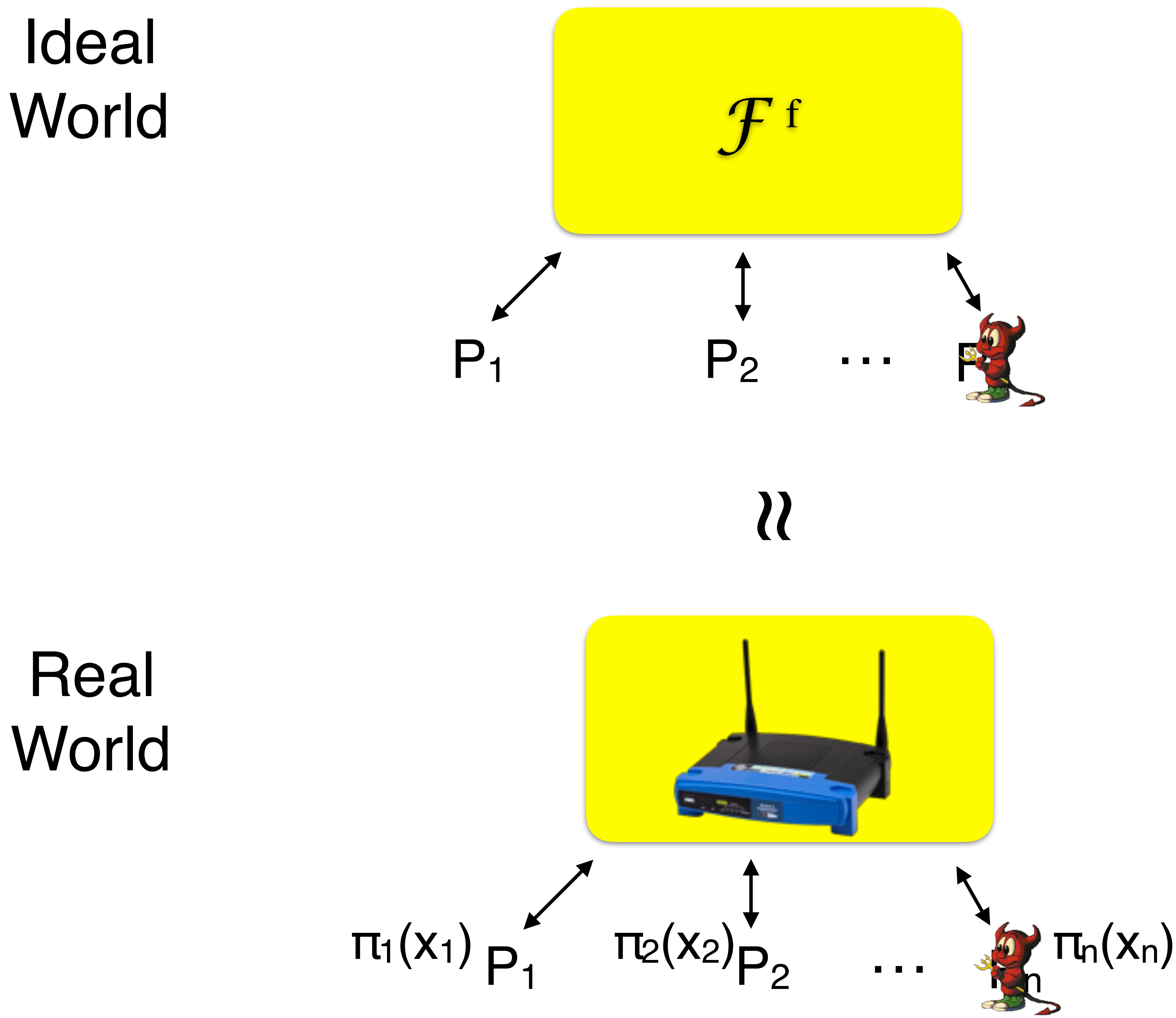
Crypto On Blockchain

Outline

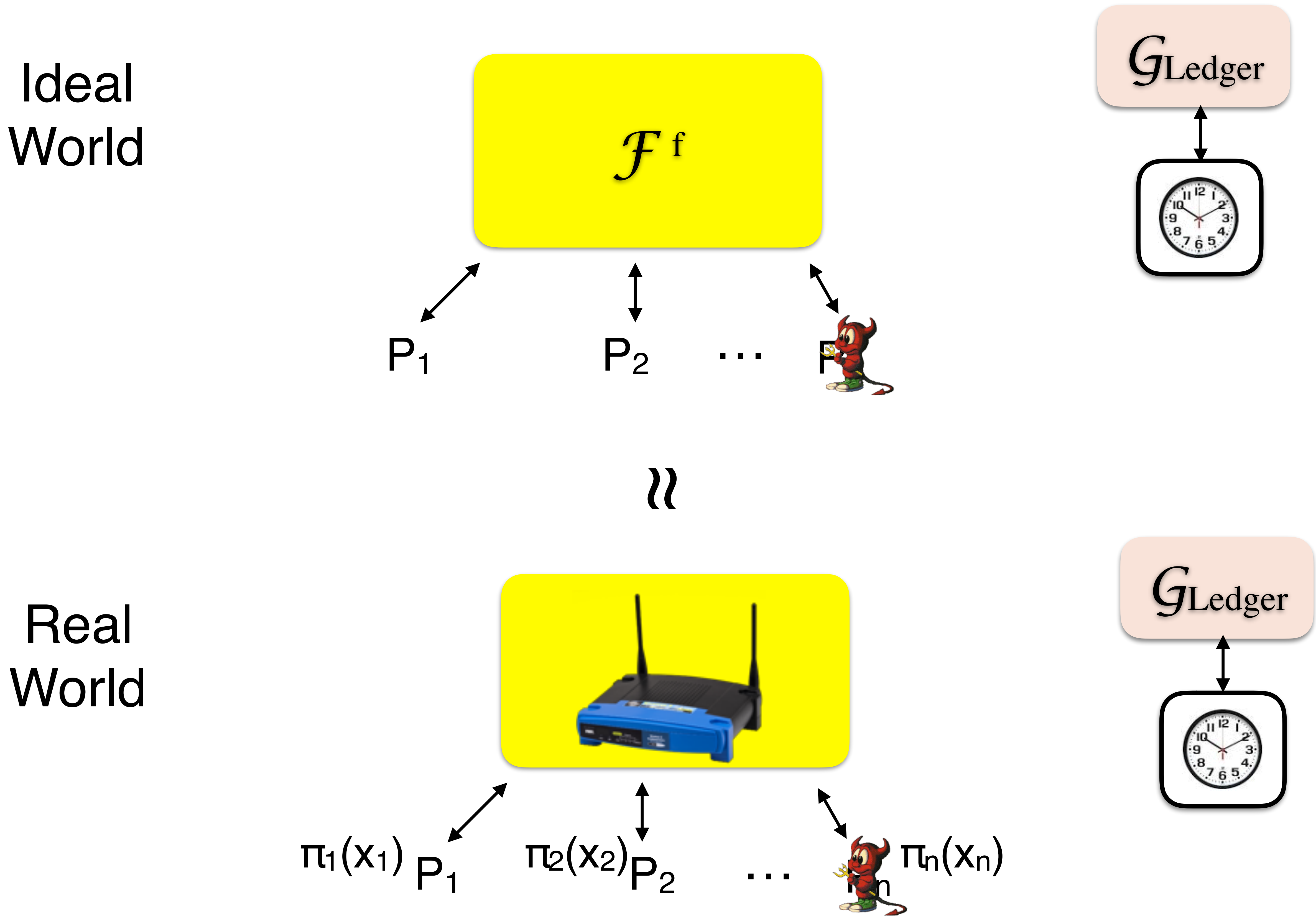
- The functionality offered by blockchains
- Leveraging Security Loss with Coins

  … in Secure Function Evaluation (SFE)
- A formal cryptographic (UC) model for security proofs

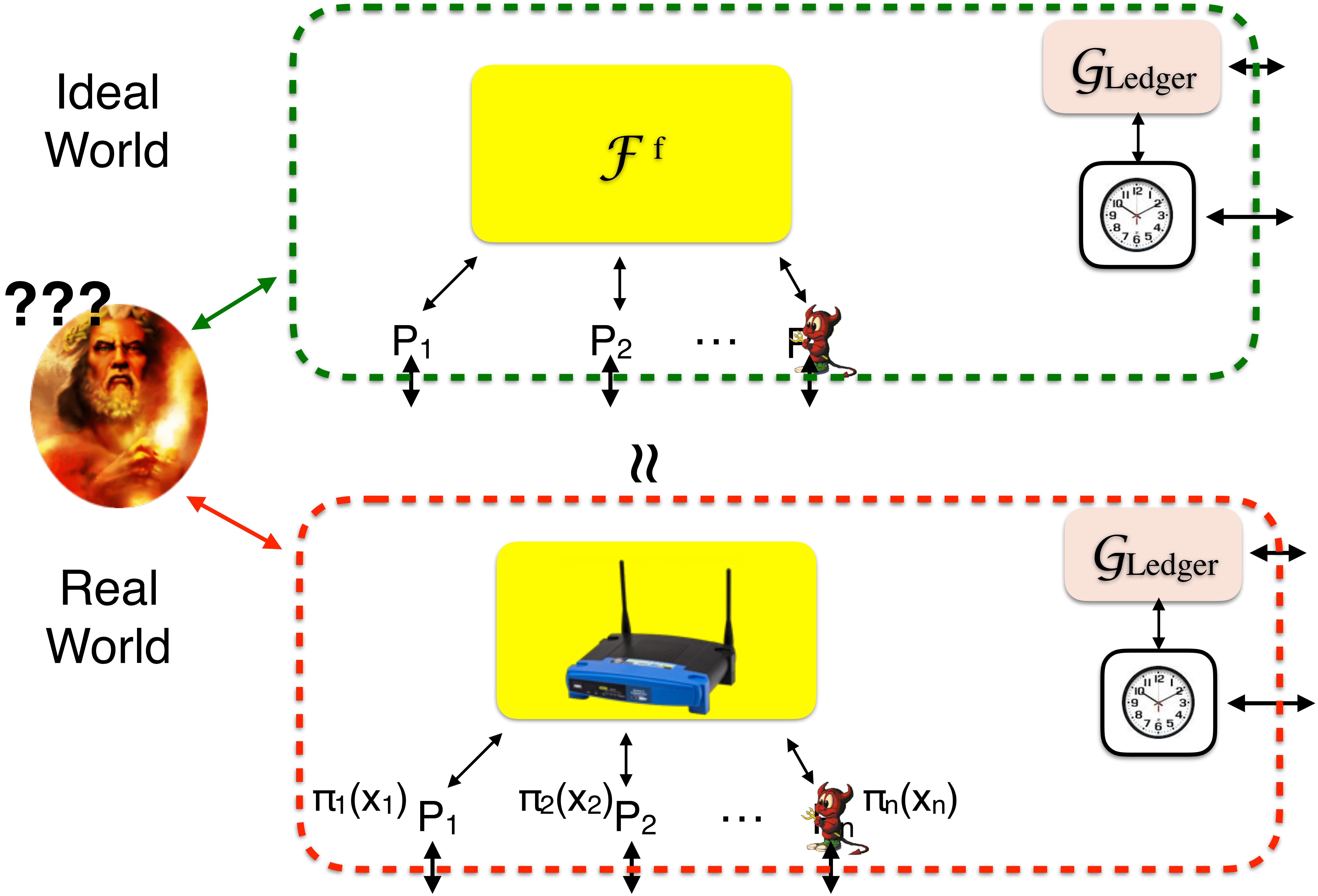
A Formal Model: GUC



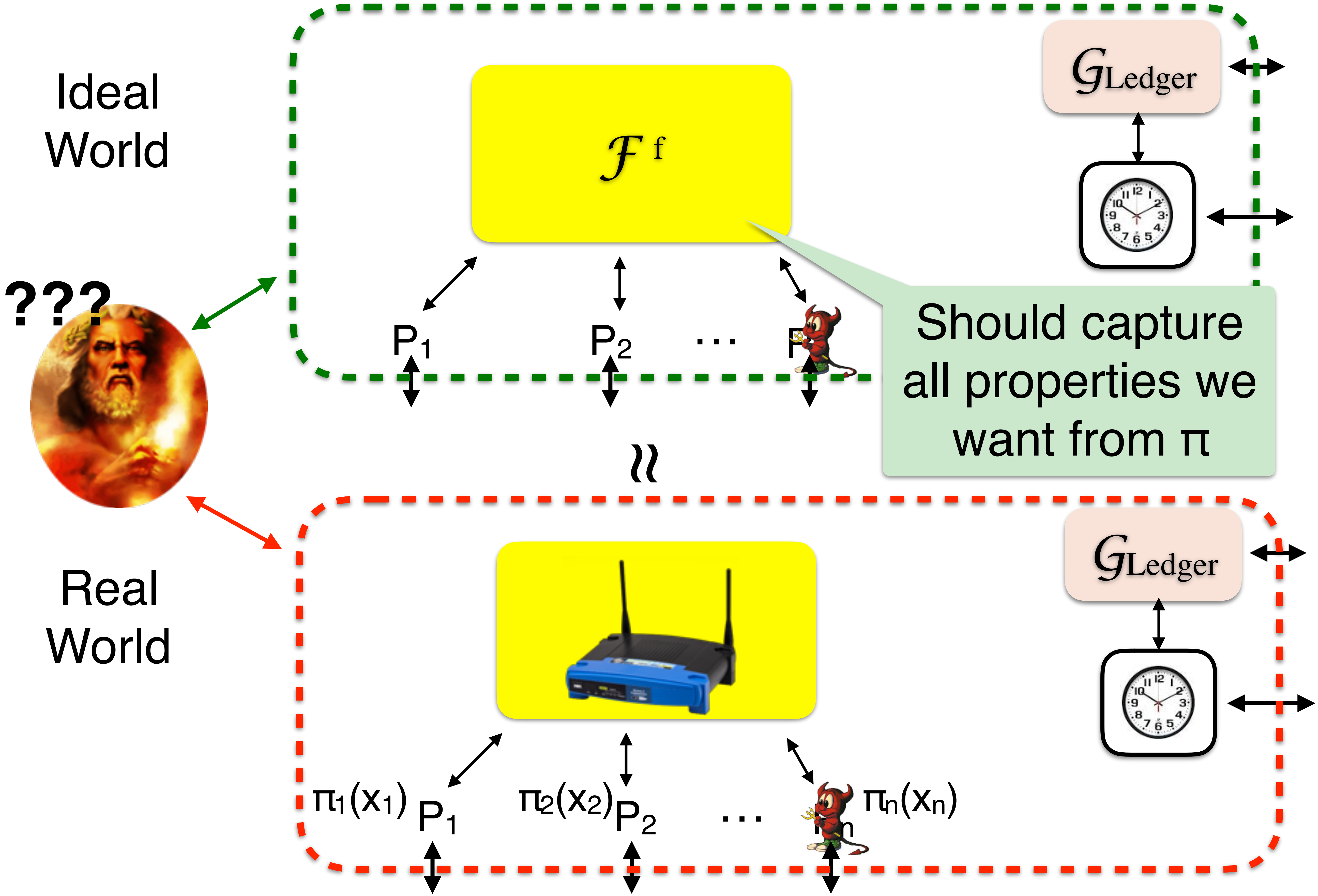
A Formal Model: GUC



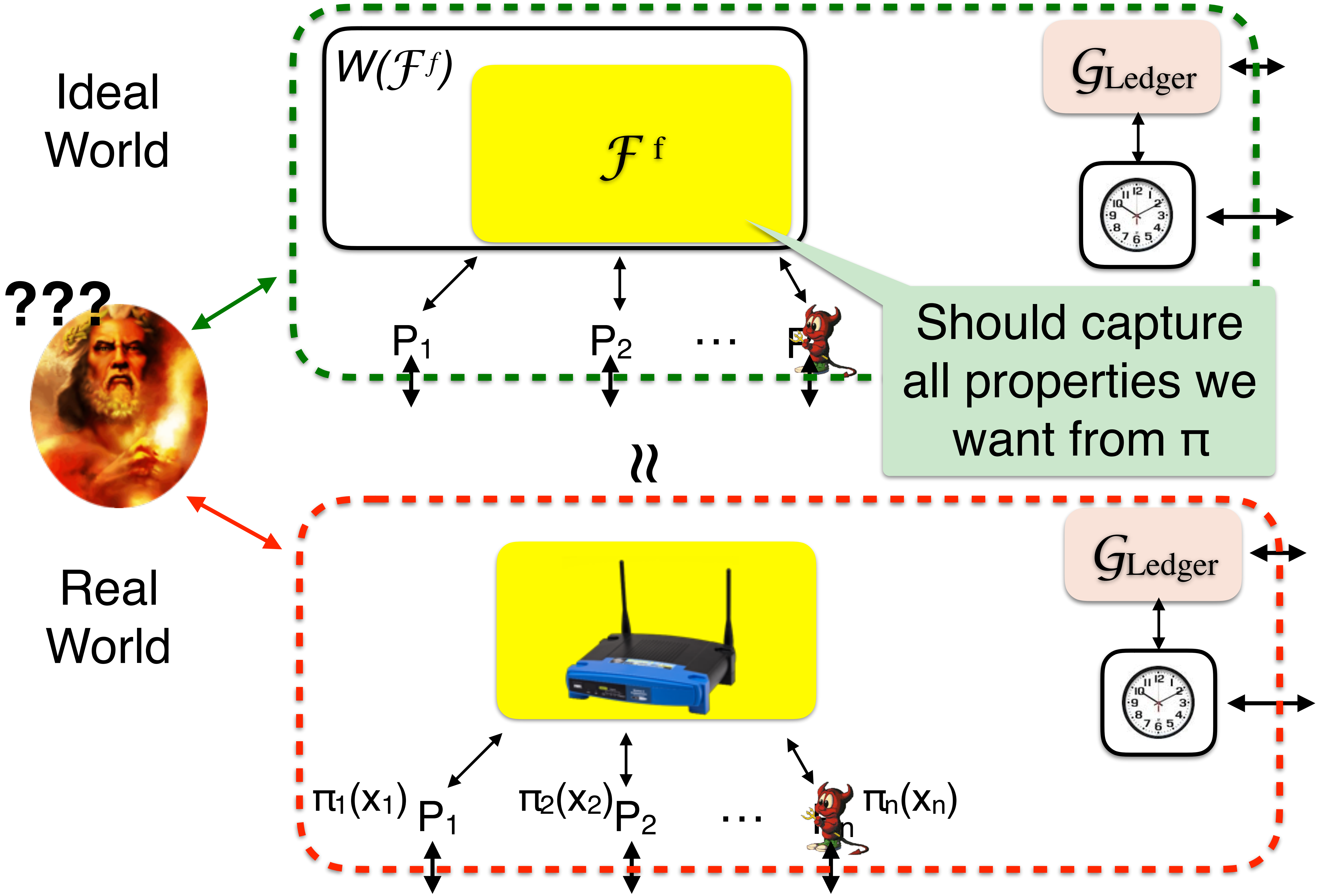
A Formal Model: GUC



A Formal Model: GUC



A Formal Model: GUC

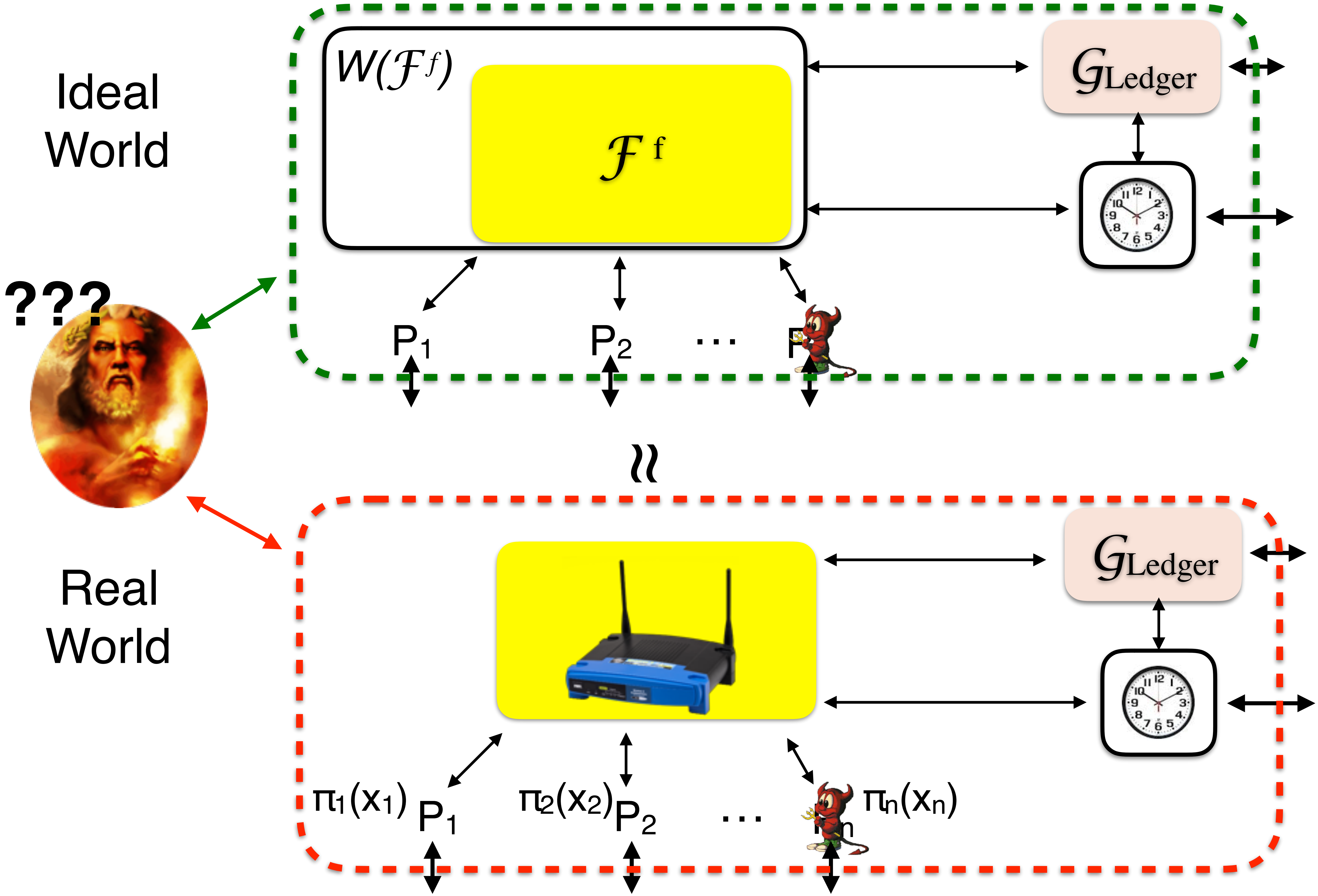


Benefits of this Modeling

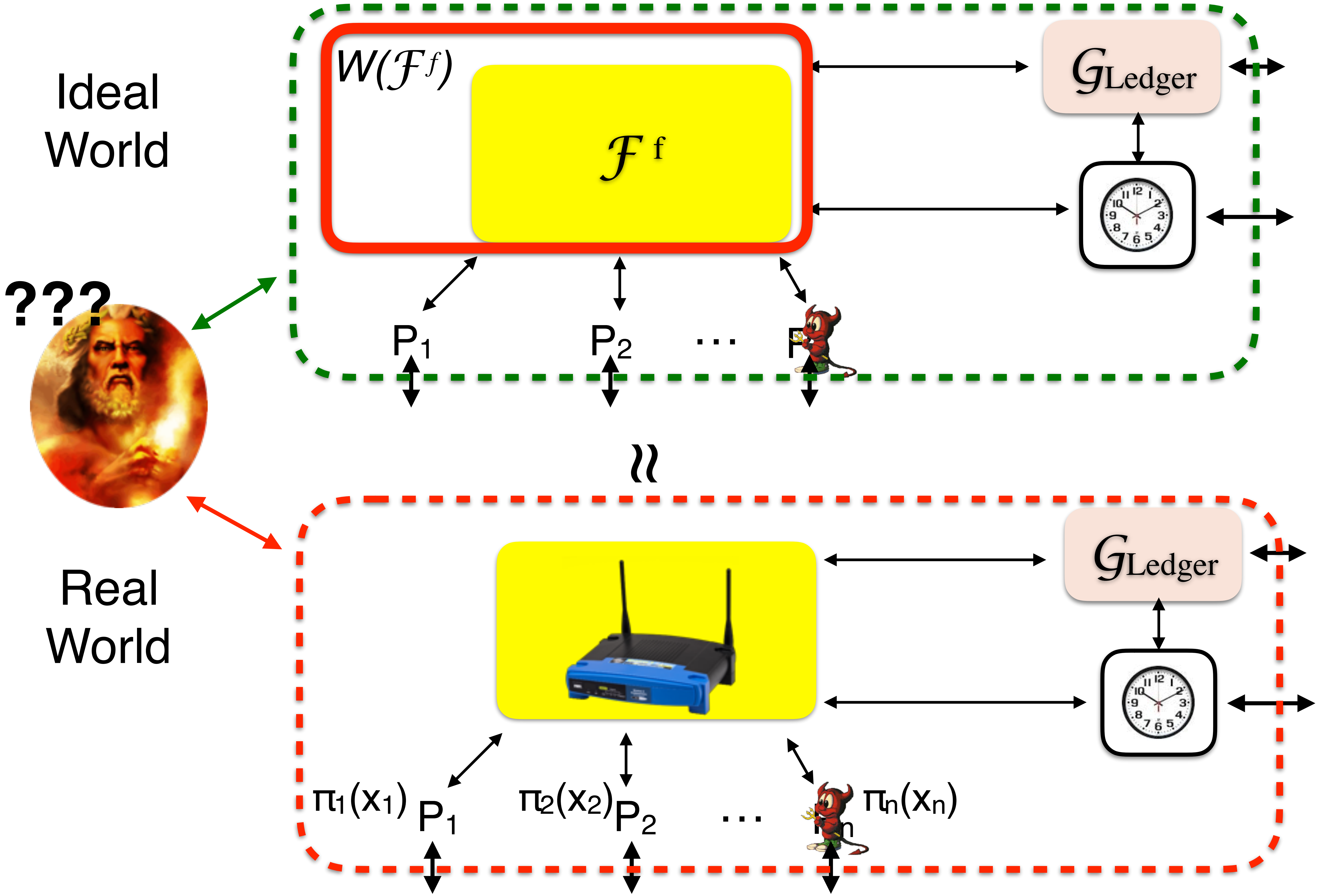
Benefits of this Modeling

- A single abstraction of the functionality offered by cryptocurrencies
  - Advanced transactions correspond to an advanced validation predicate
- A definition of *fair compensation* as a (UC) functionality-wrapper forces us to be precise
  - An explicit formation of synchrony with a single global clock (capturing what protocols assume in reality).
- Compatibility with standard (formal) analysis of crypto protocols
- A (universal) composition theorem

A Formal Model: GUC



A Formal Model: GUC



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$

SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$

Idea: The predicates are used to **filter** the adversarial influence

- $Q^{Init}(State, Wallet_i) = True$ iff the $Wallet_i$ has enough funds
- $Q^{Dlvr}(State, Wallet_i) = True$ iff it is OK to deliver to P_i
  - E.g., if P_i does not “owe” money
- $Q^{Abrt}(State, Wallet_i) = True$ iff it is OK for P_i to abort
  - E.g., if P_i has an increase of funds

SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$

$\mathcal{G}_{ledger}$

$W(\mathcal{F}^f)$

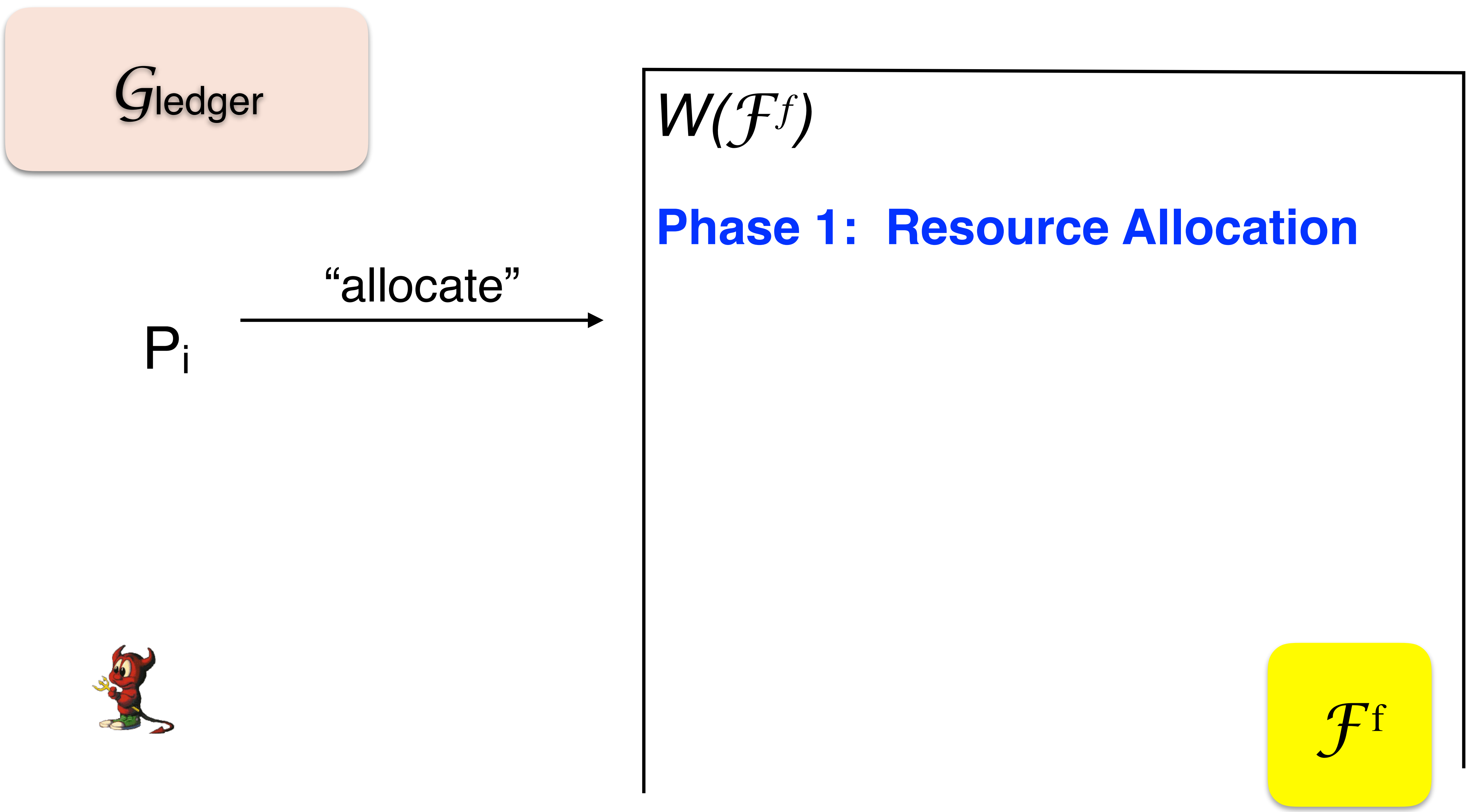
Phase 1: Resource Allocation

$\mathcal{F}^{f}$

SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

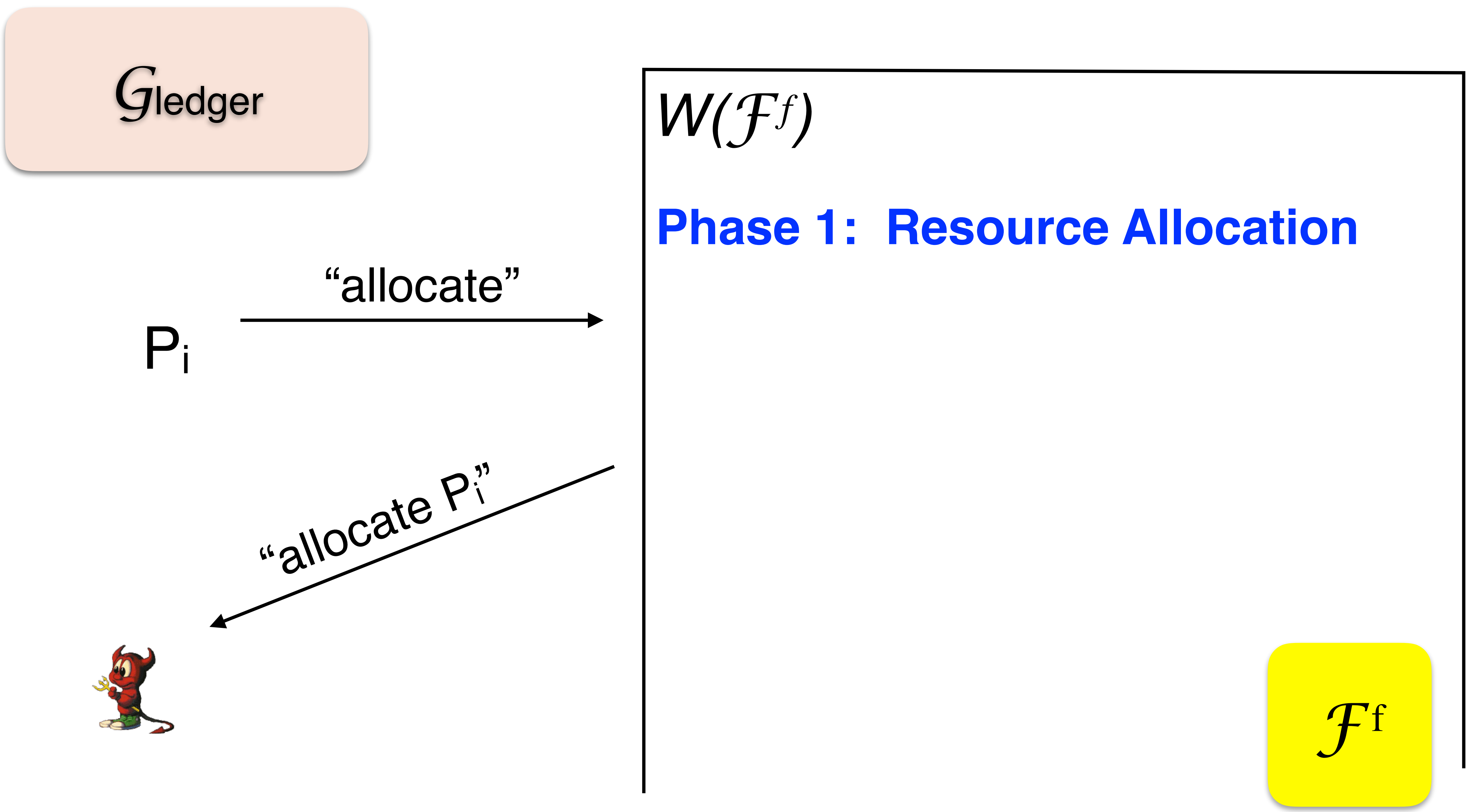
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

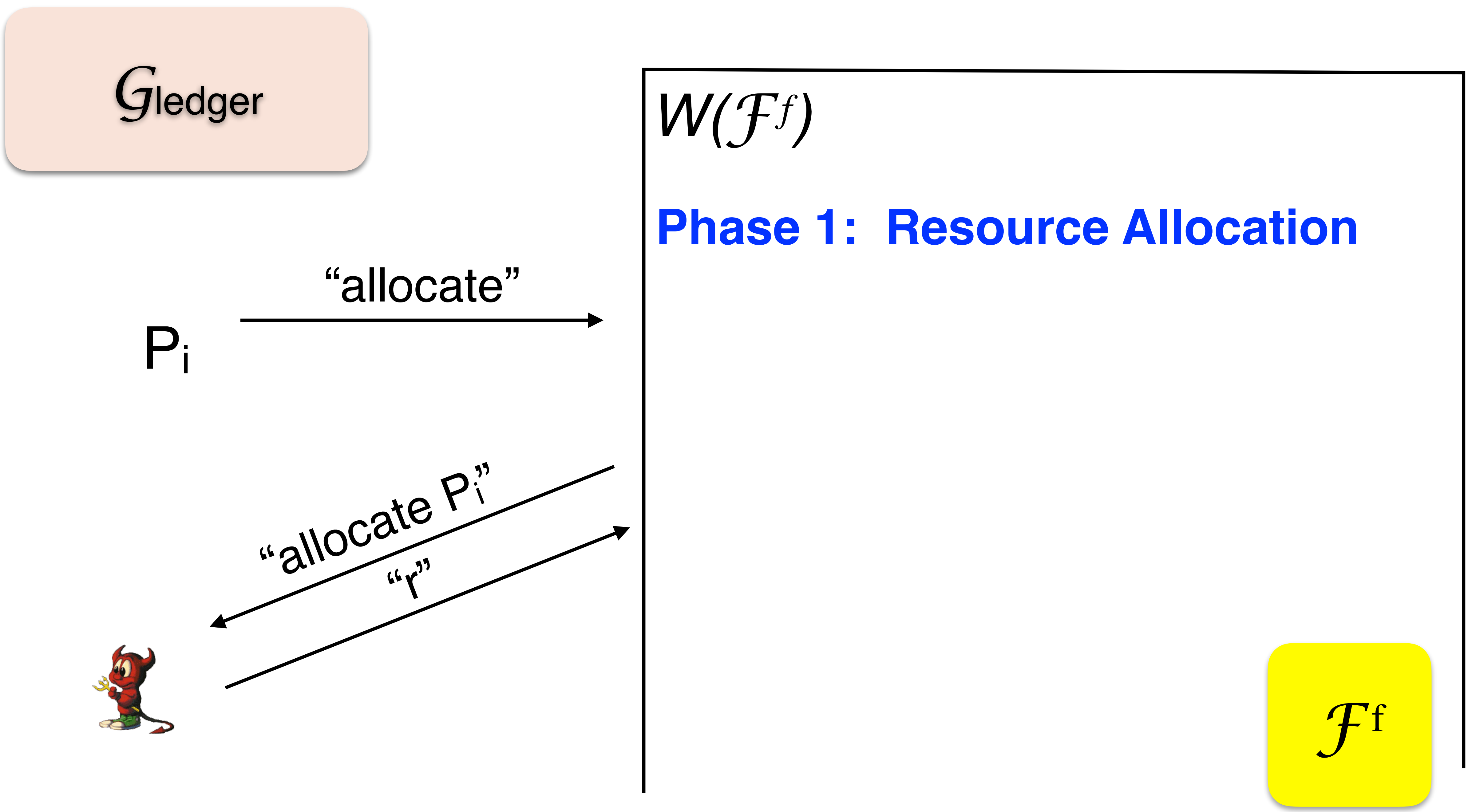
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

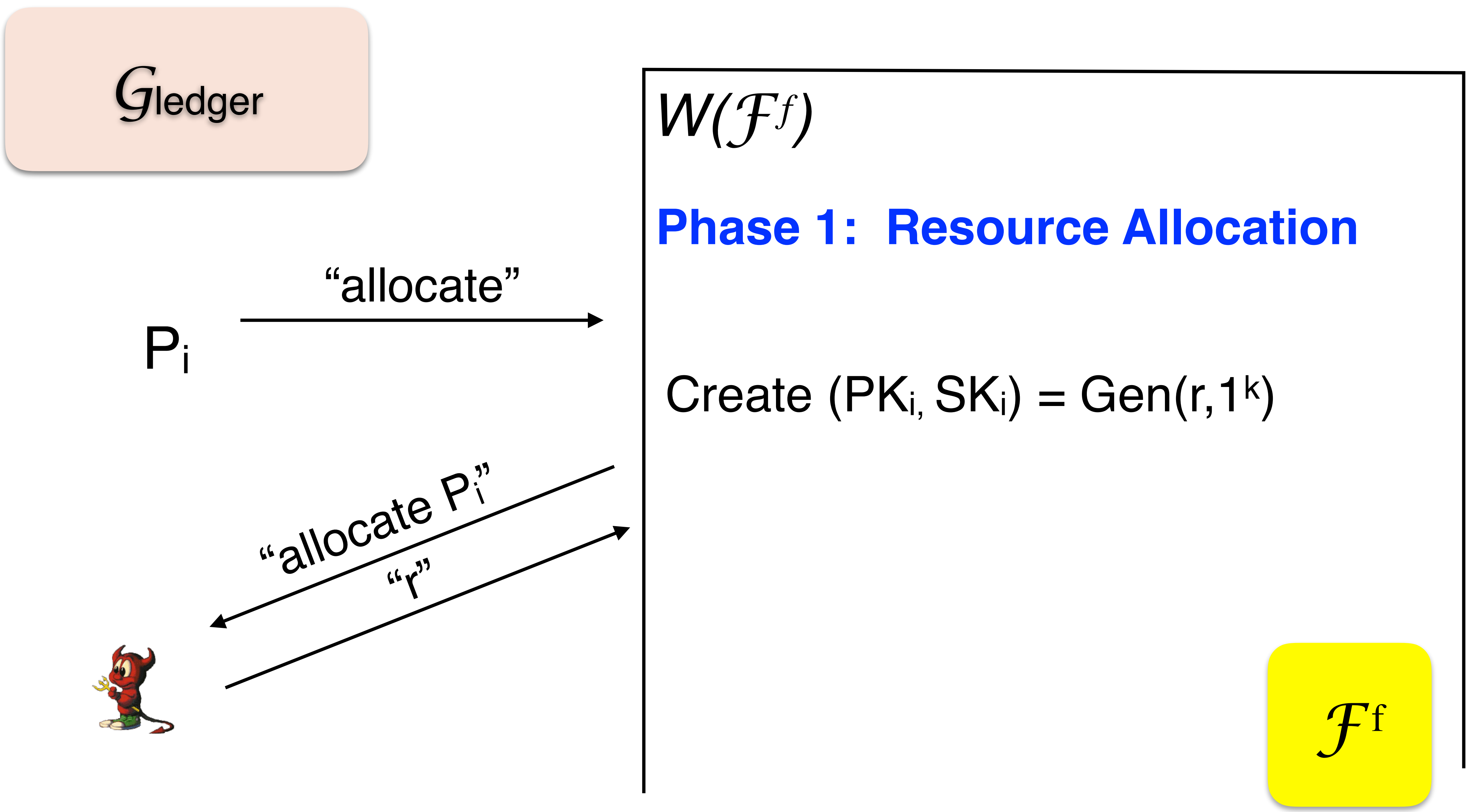
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

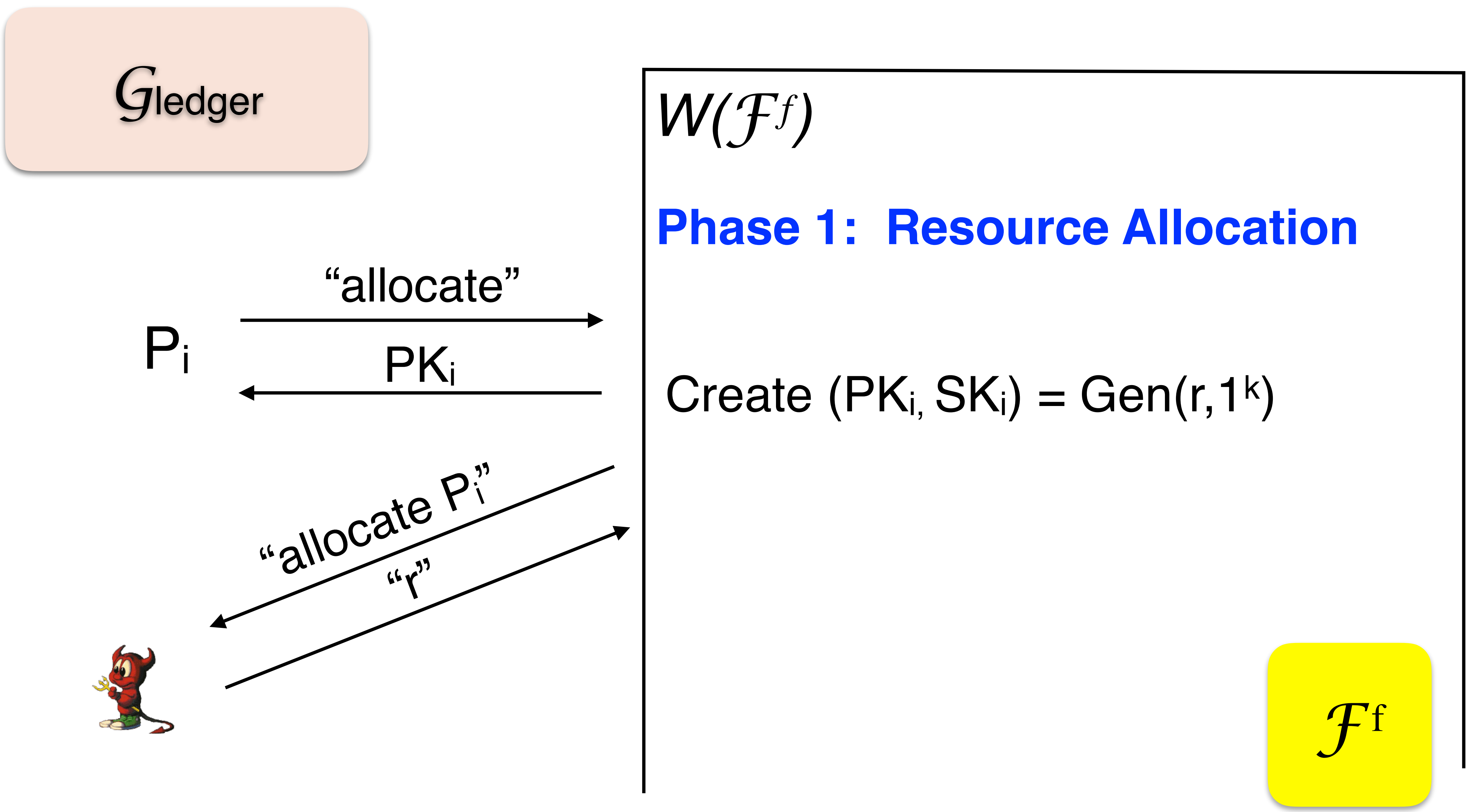
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

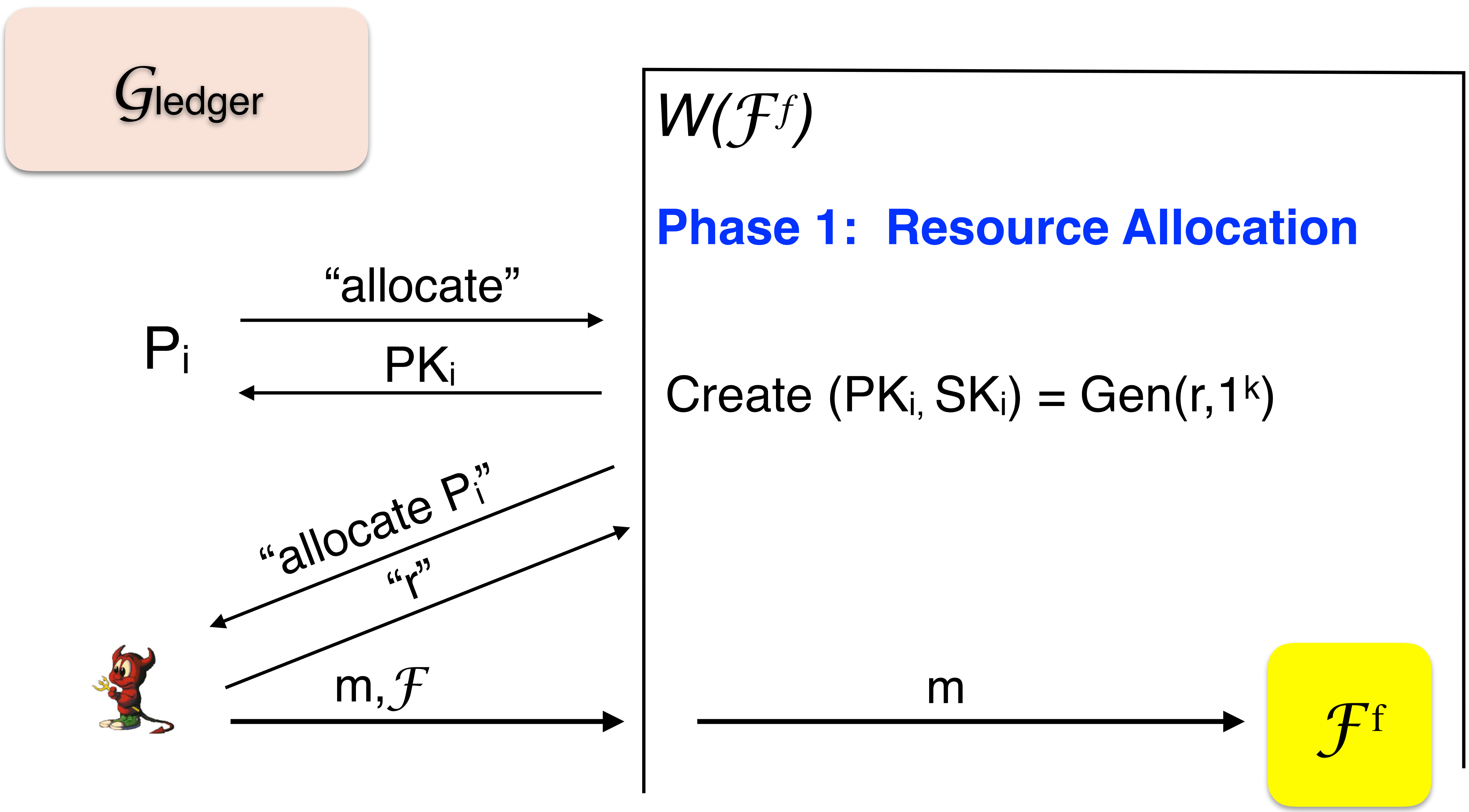
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

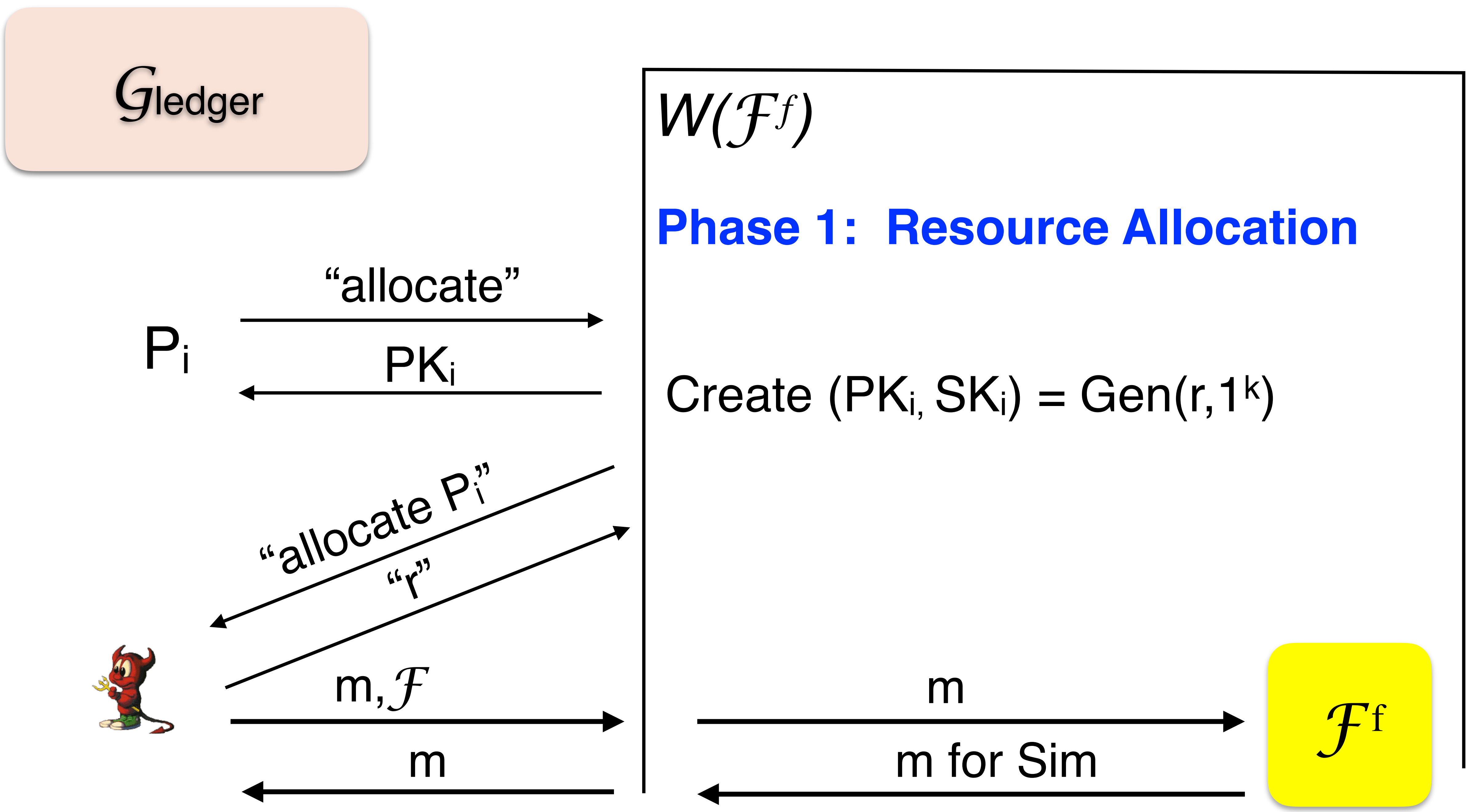
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

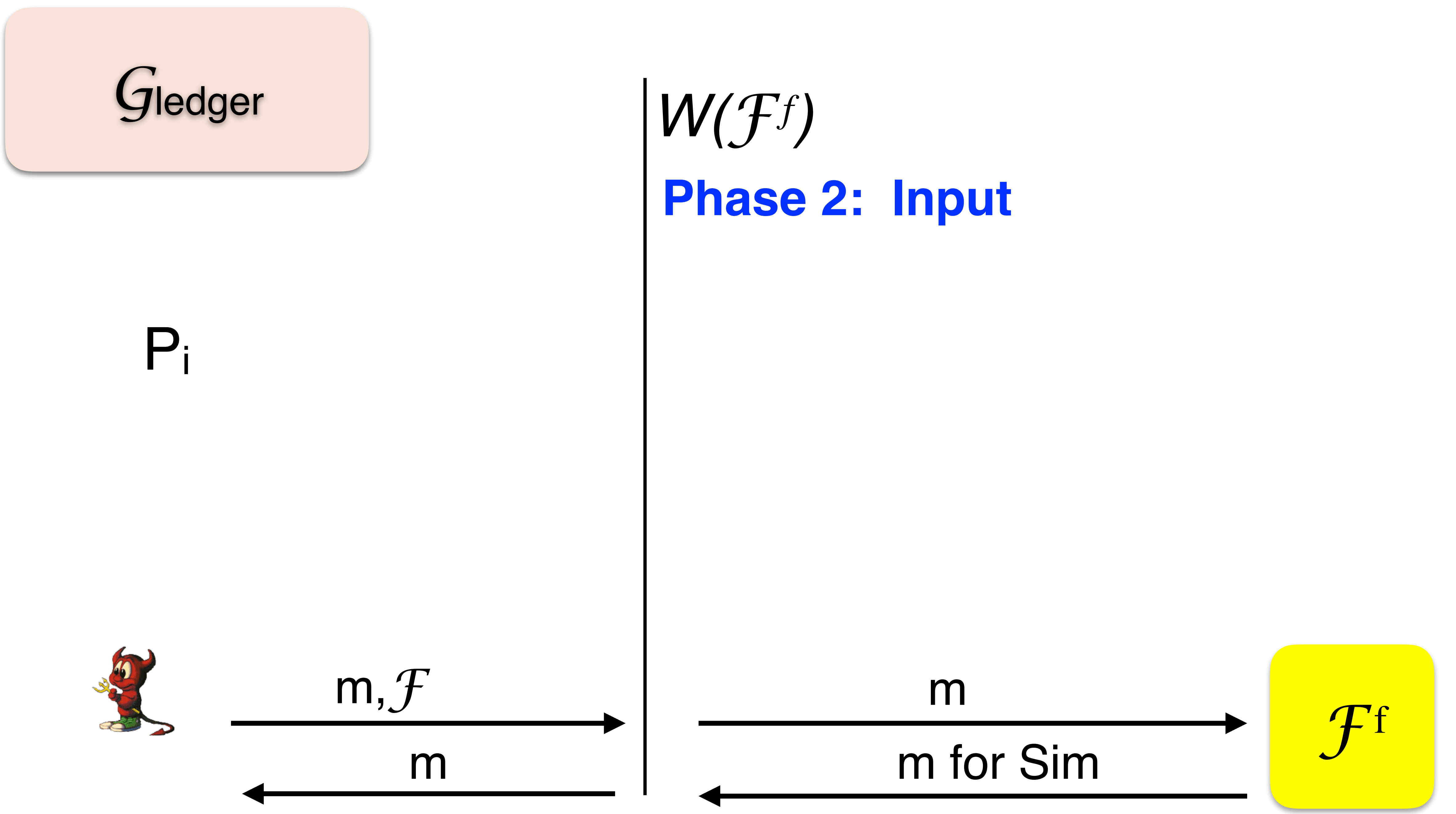
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

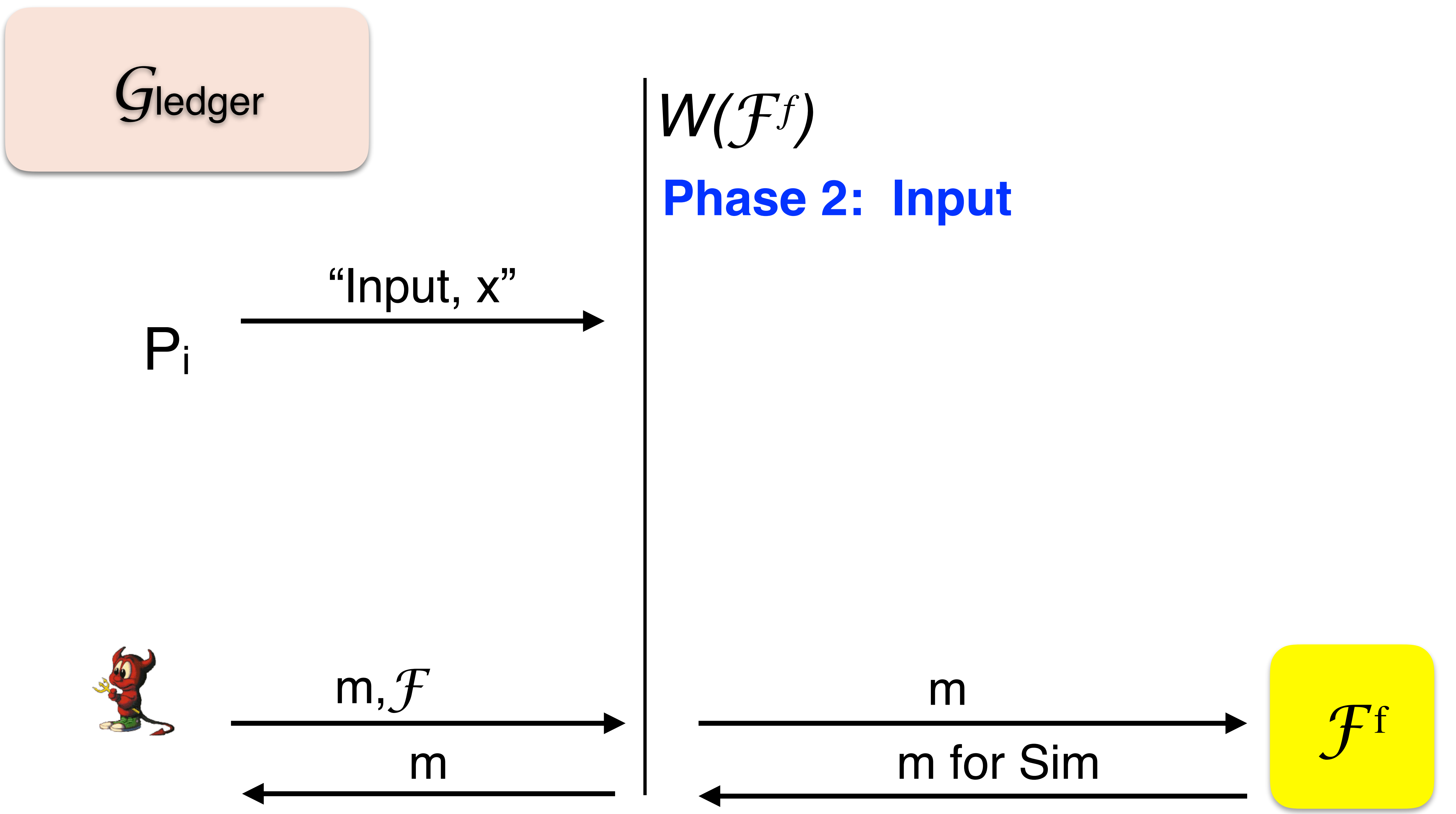
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

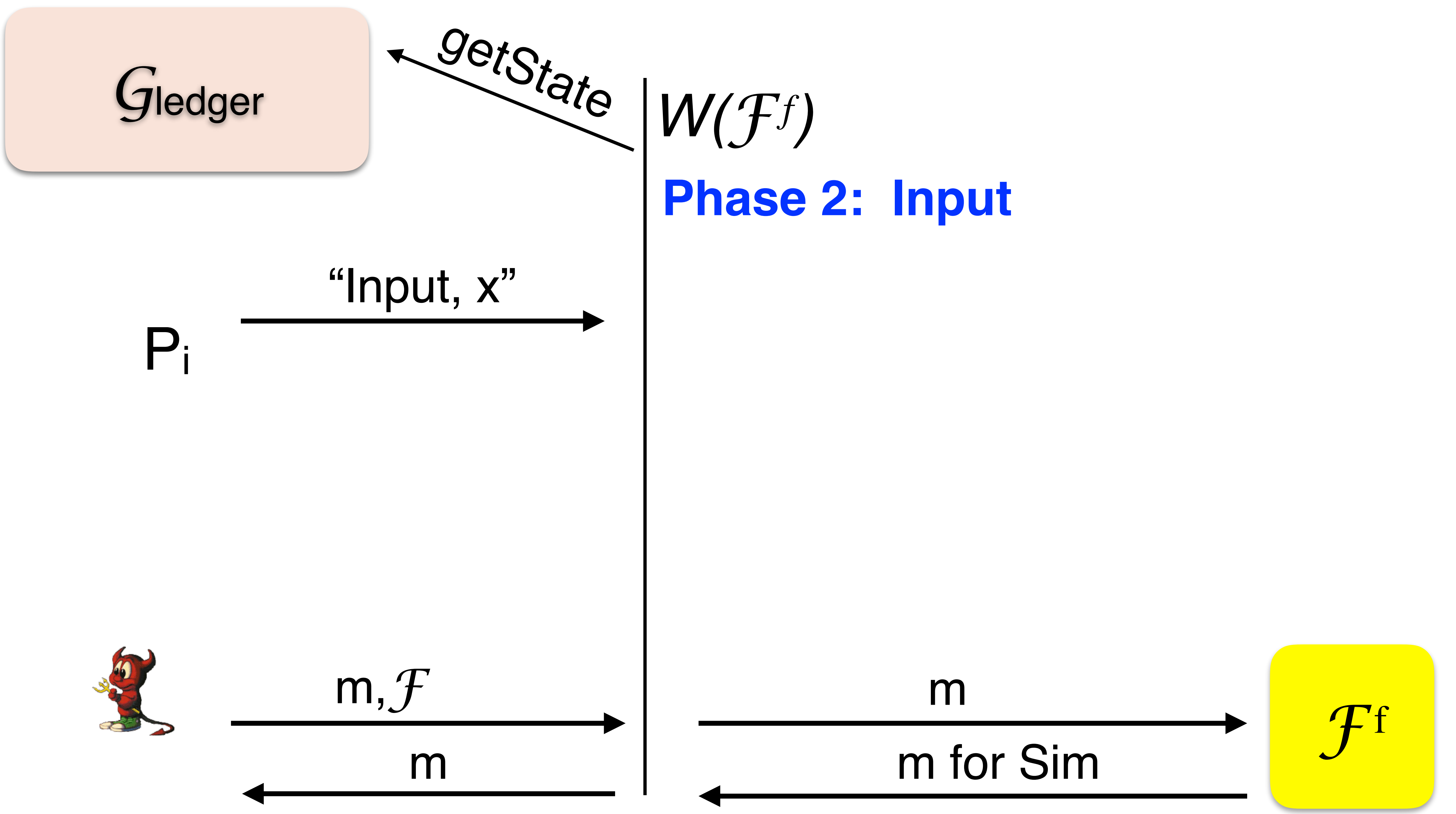
- (Q^{Init}, Q^{Dlvr}, Q^{Abrt})



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

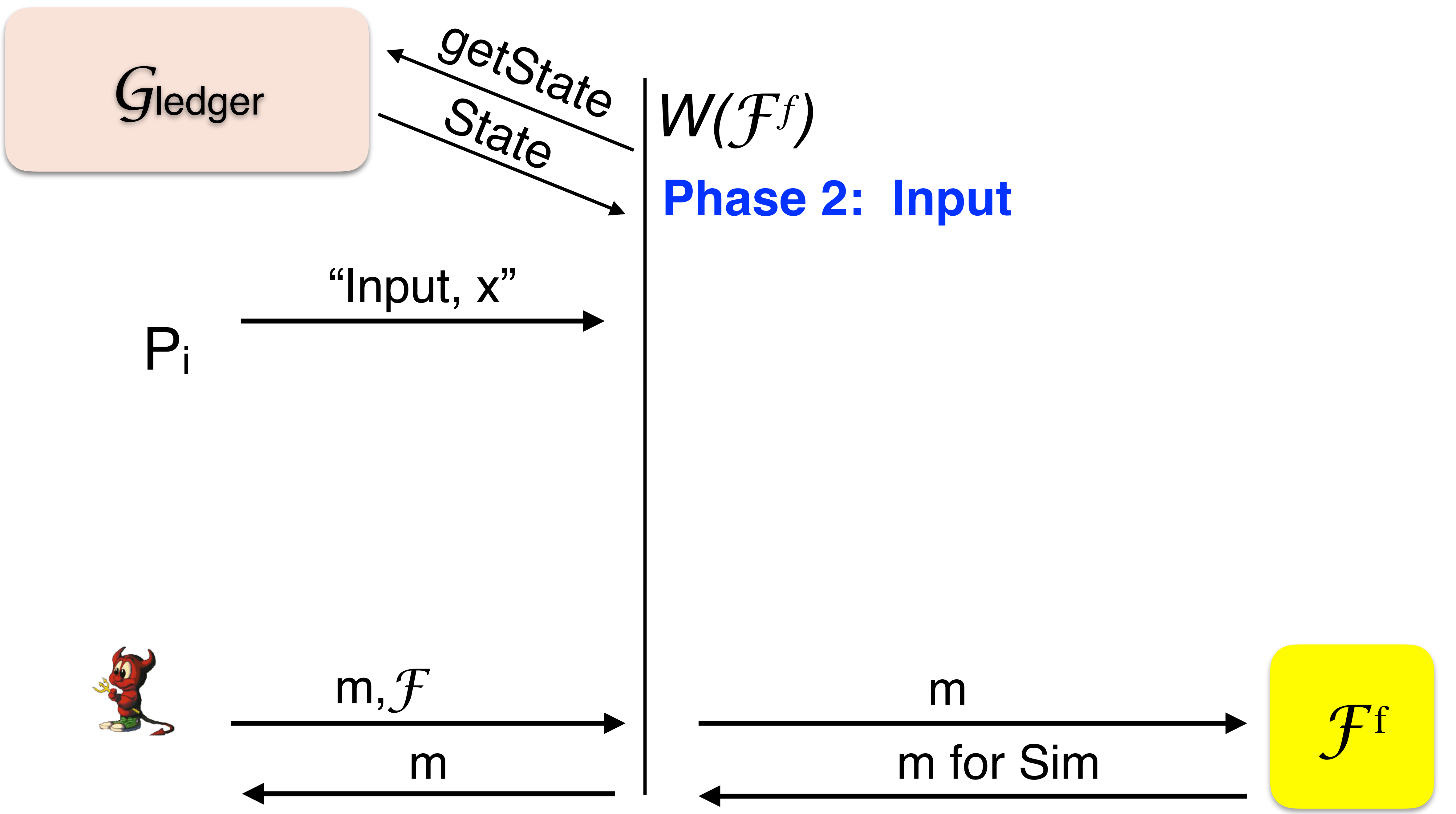
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

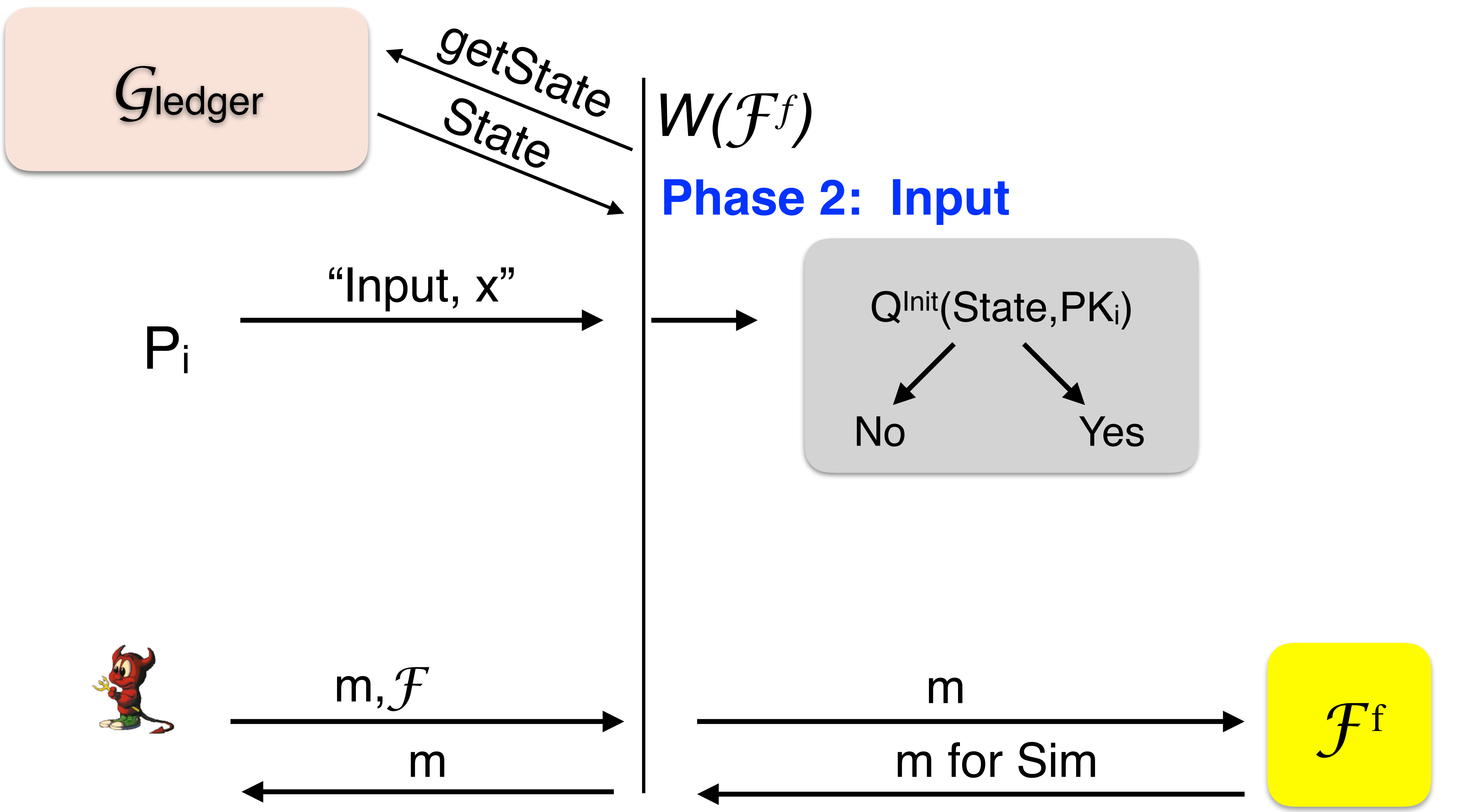
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

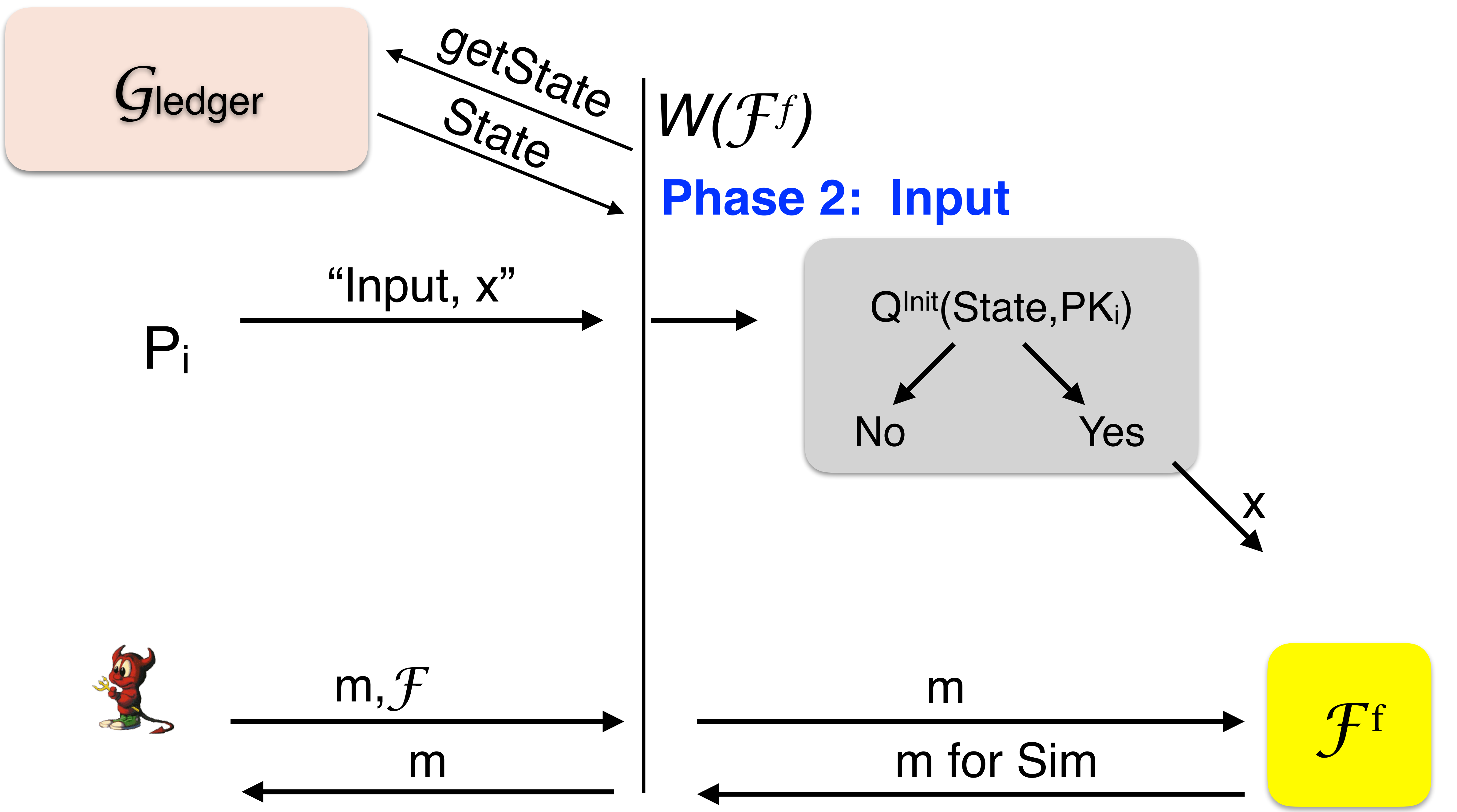
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

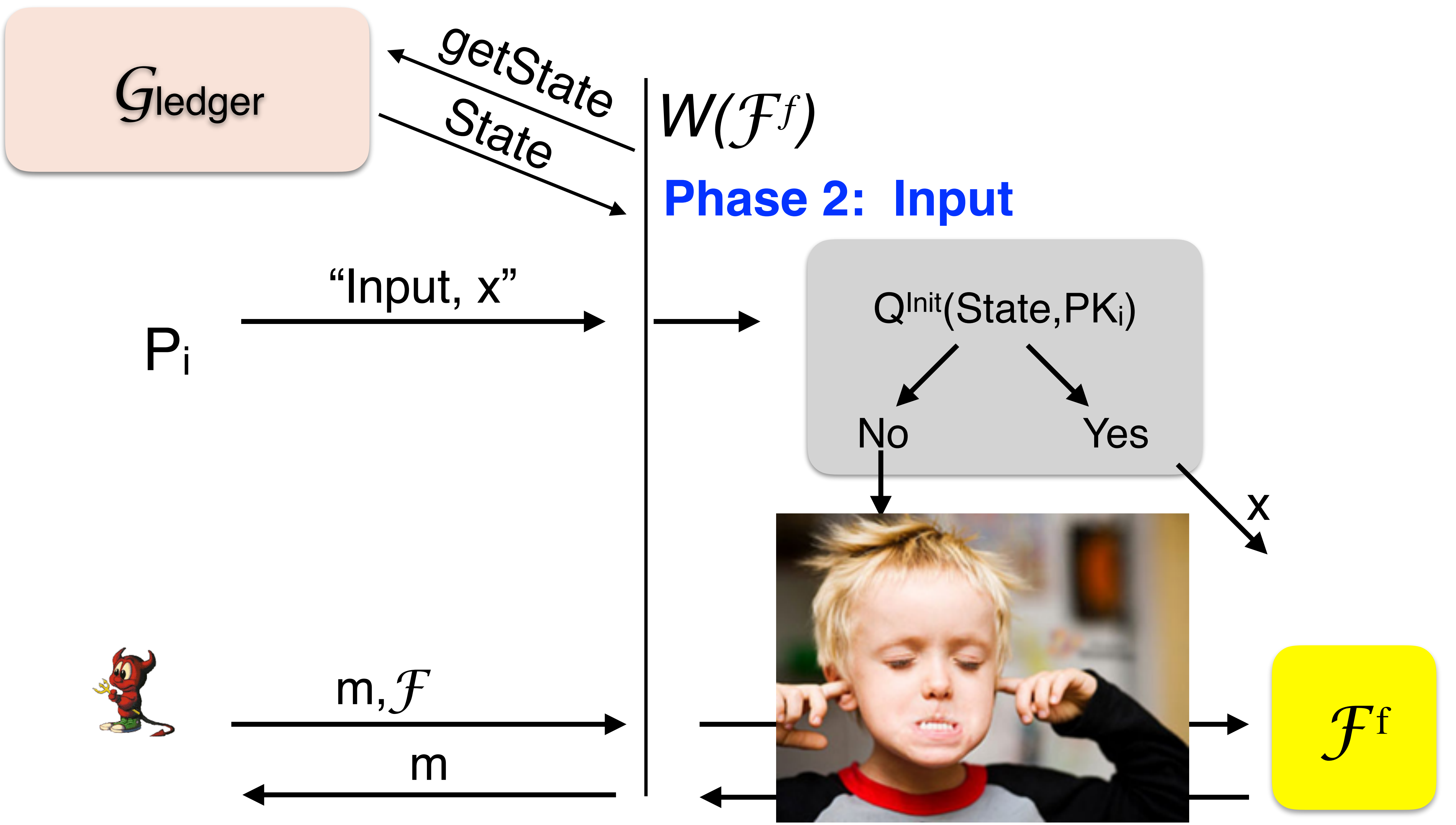
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

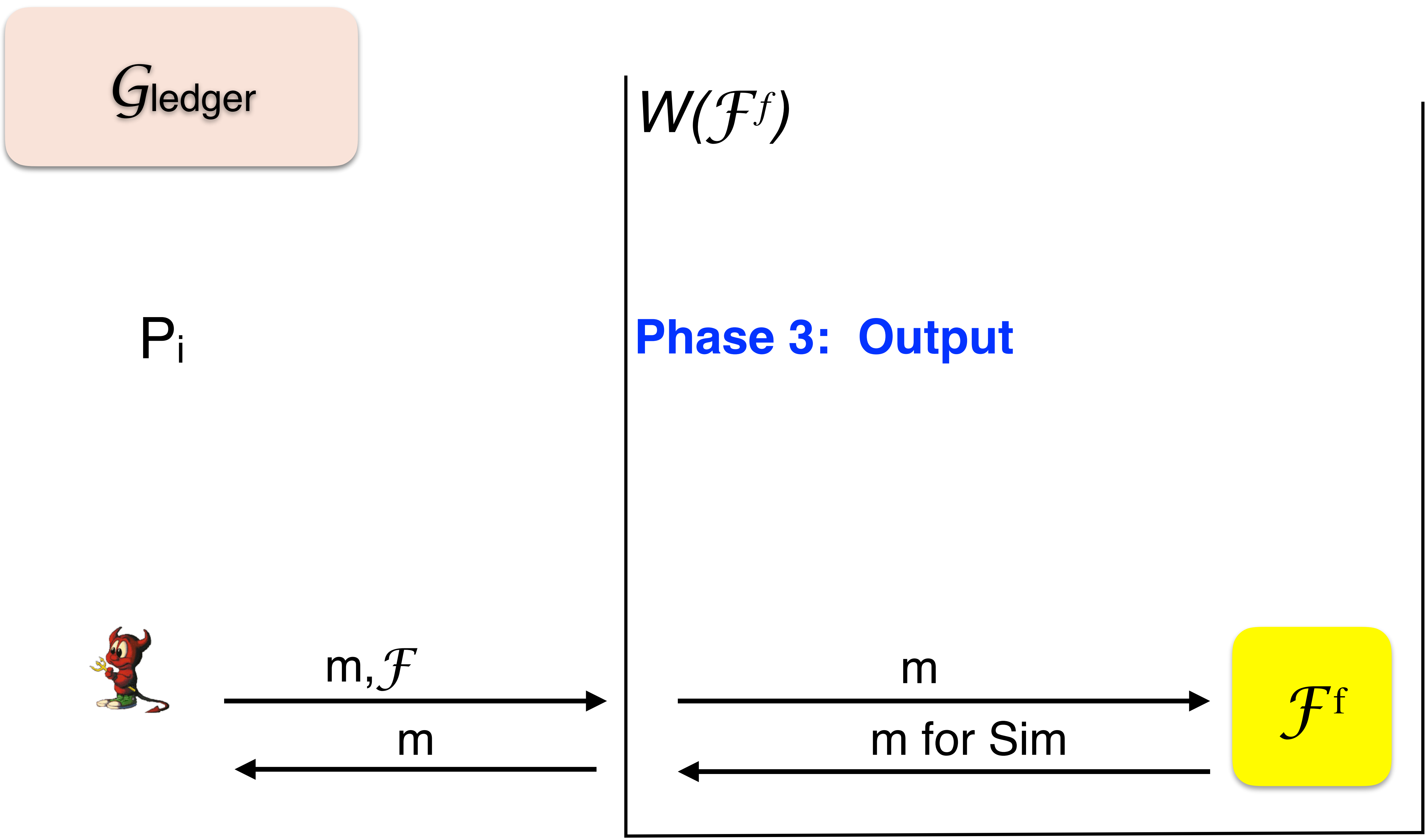
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\ldots,Pn}(\mathcal{F}^f)$ with three predicates:

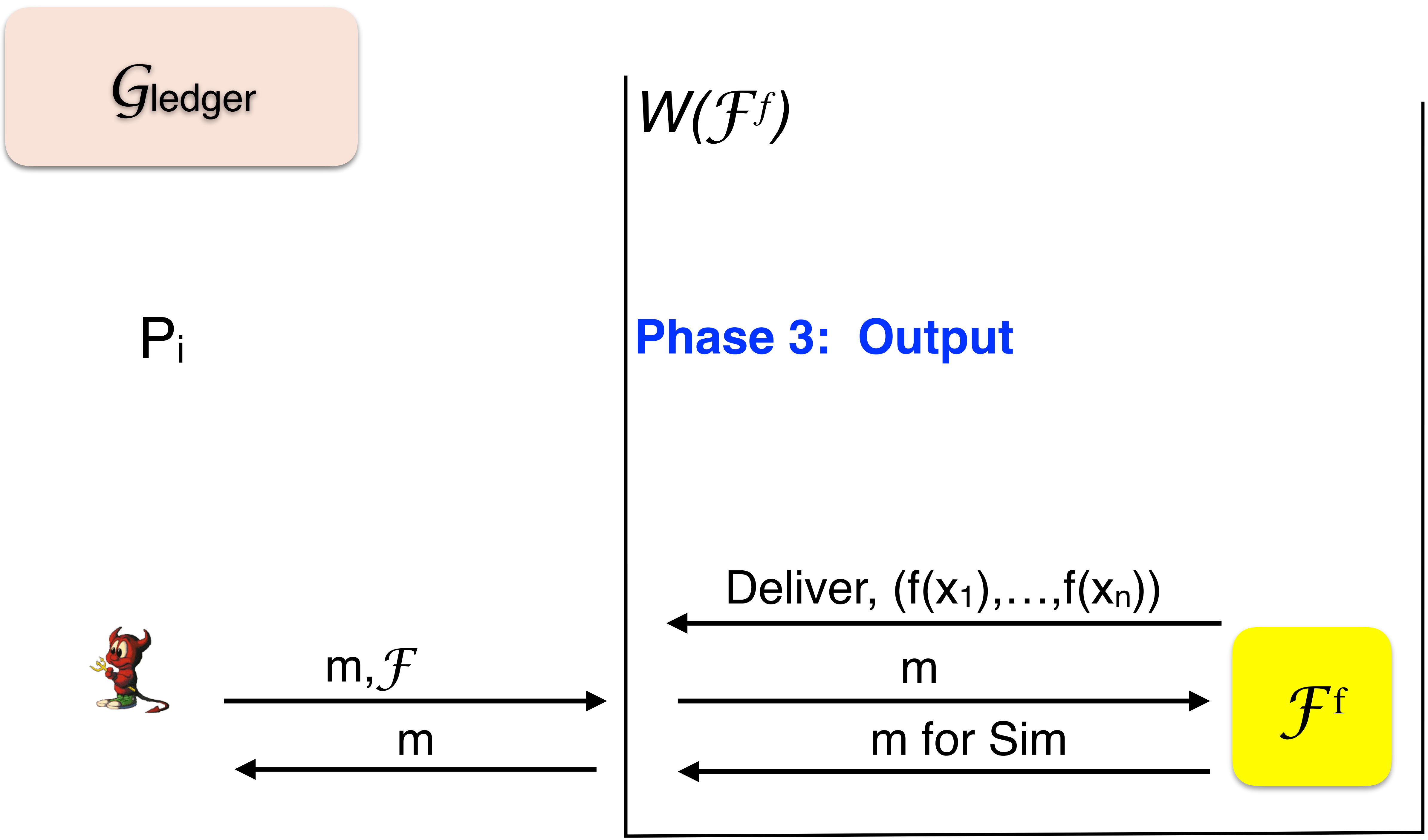
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\dots,Pn}(\mathcal{F}^f)$ with three predicates:

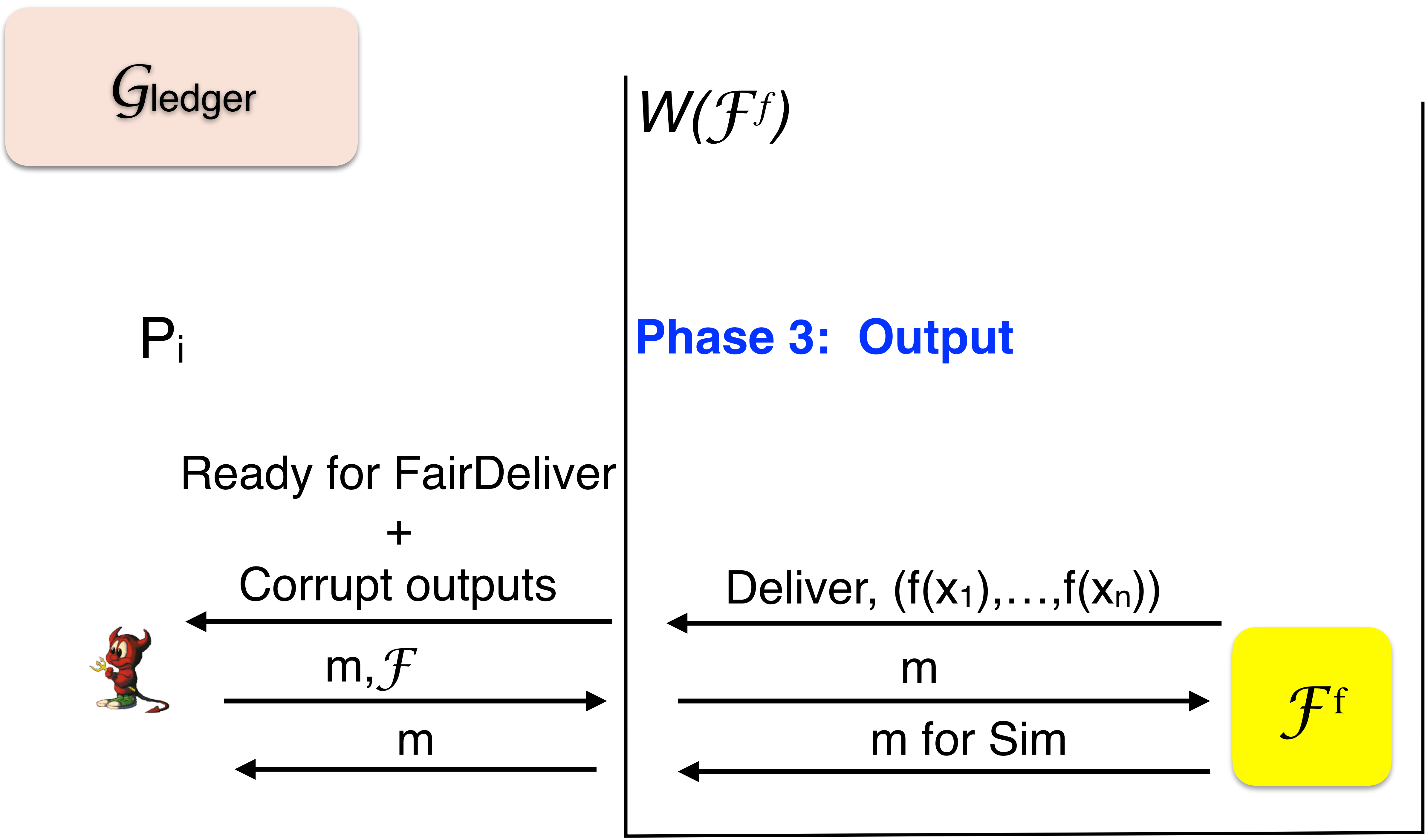
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\ldots,Pn}(\mathcal{F}^f)$ with three predicates:

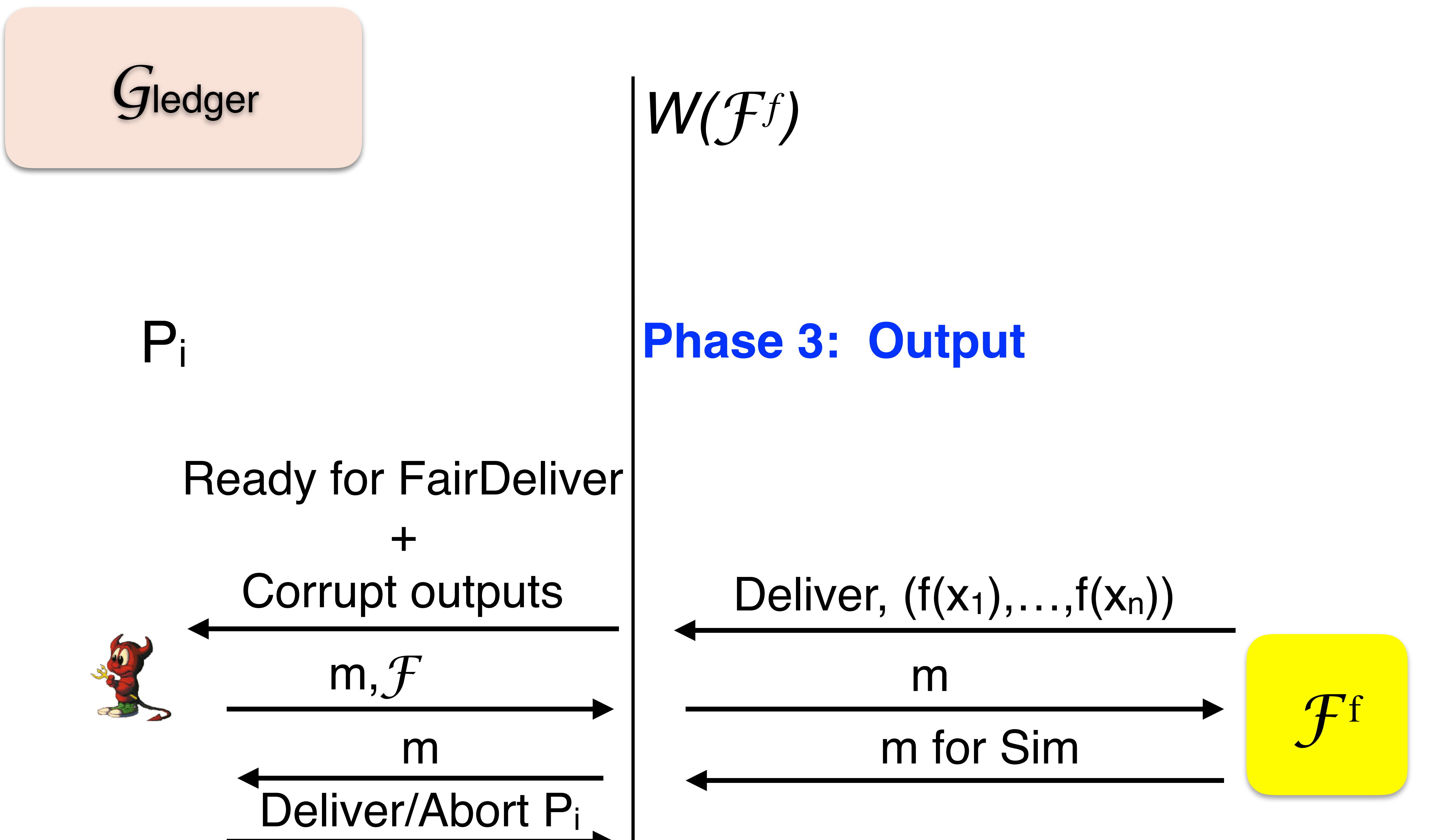
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\ldots,Pn}(\mathcal{F}^f)$ with three predicates:

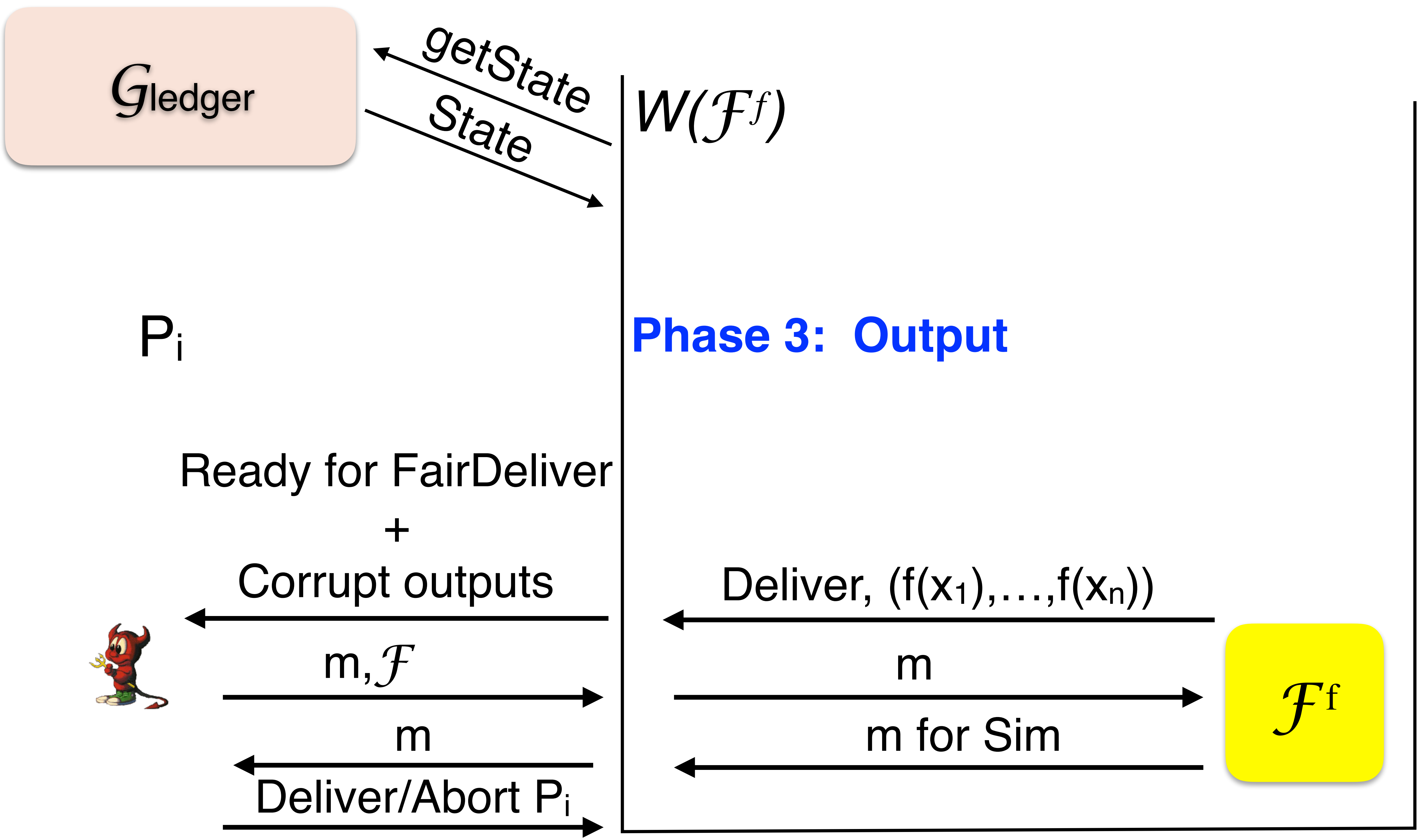
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\ldots,Pn}(\mathcal{F}^f)$ with three predicates:

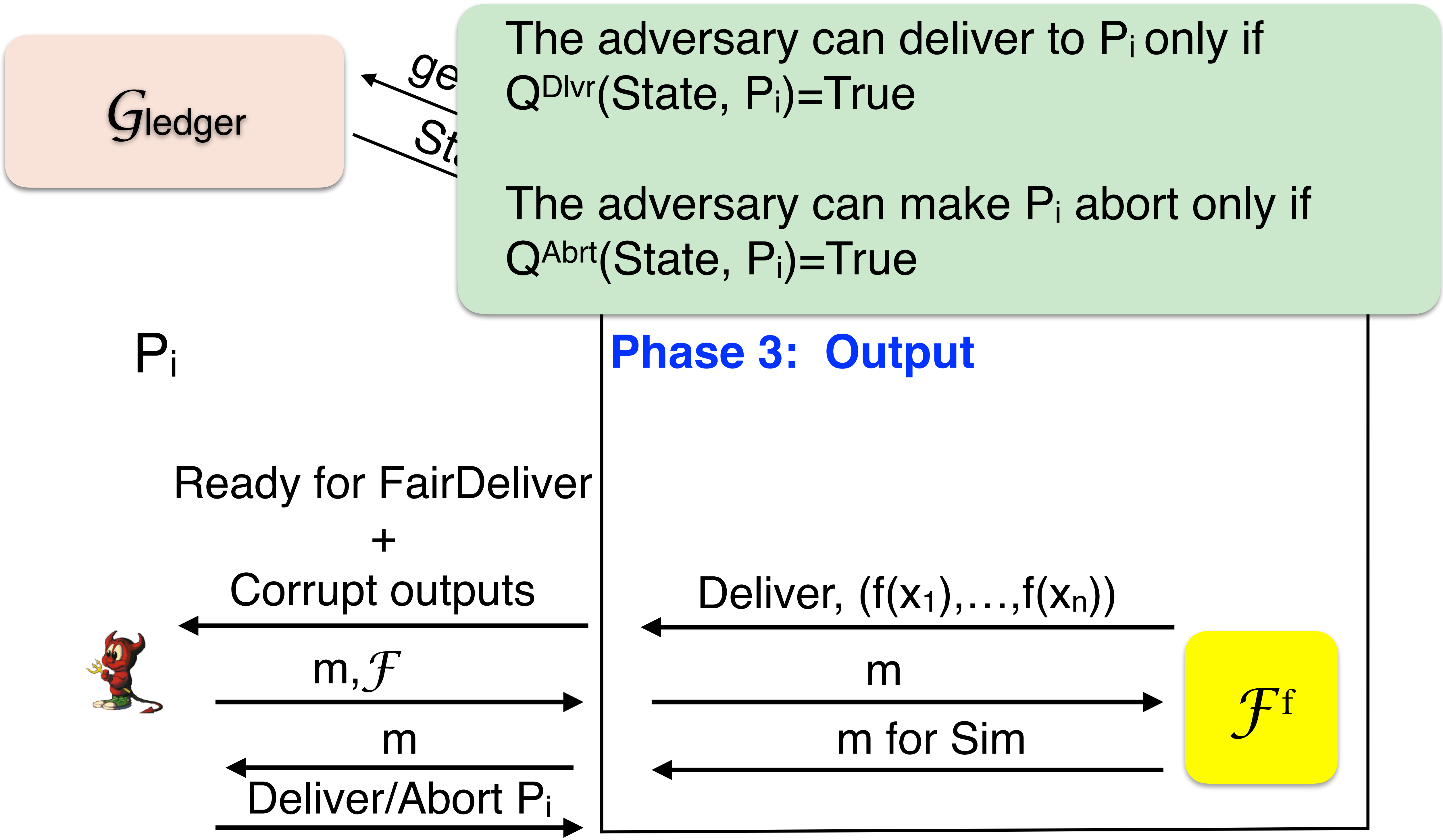
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\dots,Pn}(\mathcal{F}^{f})$ with three predicates:

- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$



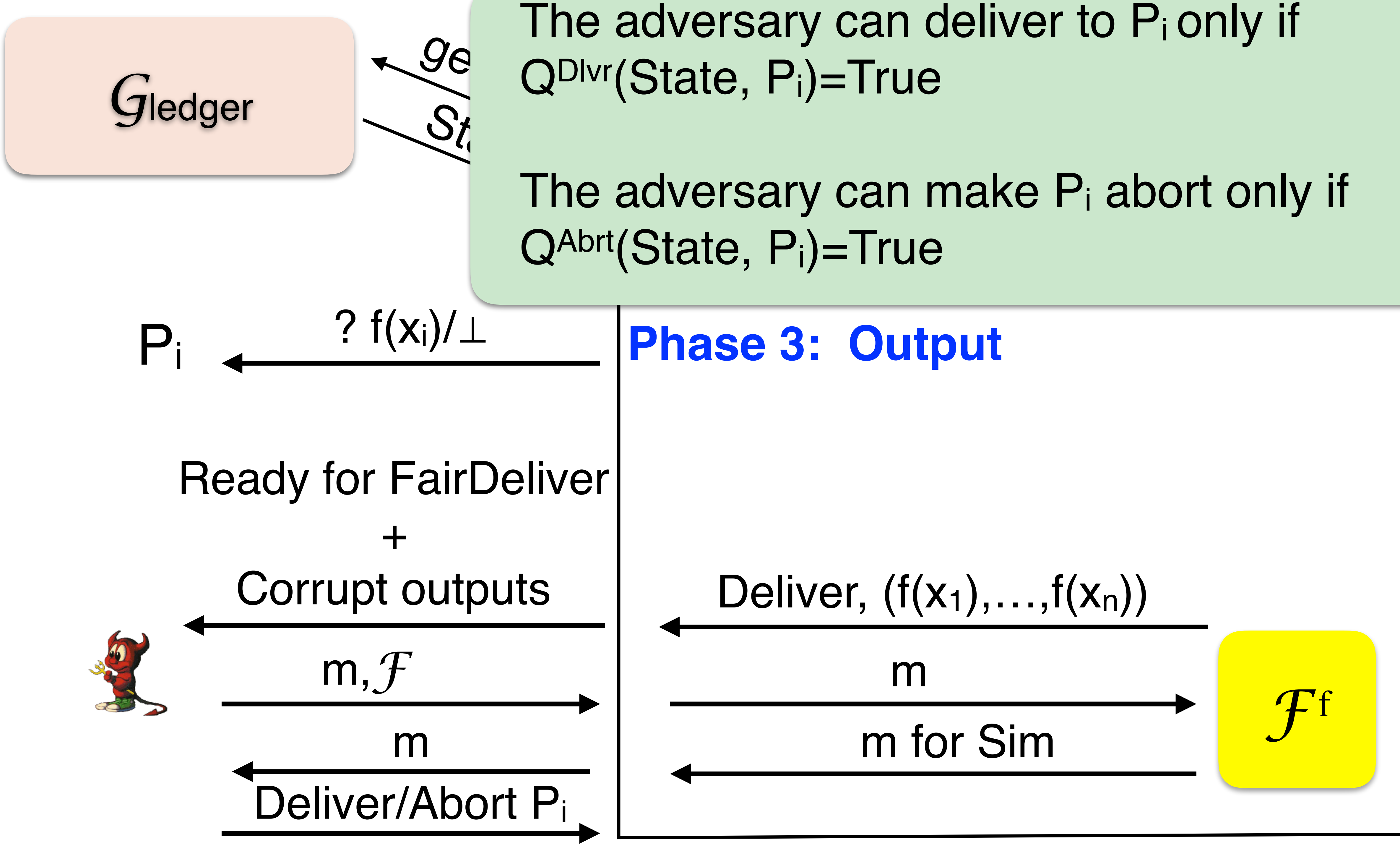
SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P1,\ldots,Pn}(\mathcal{F}^f)$ with three predicates:

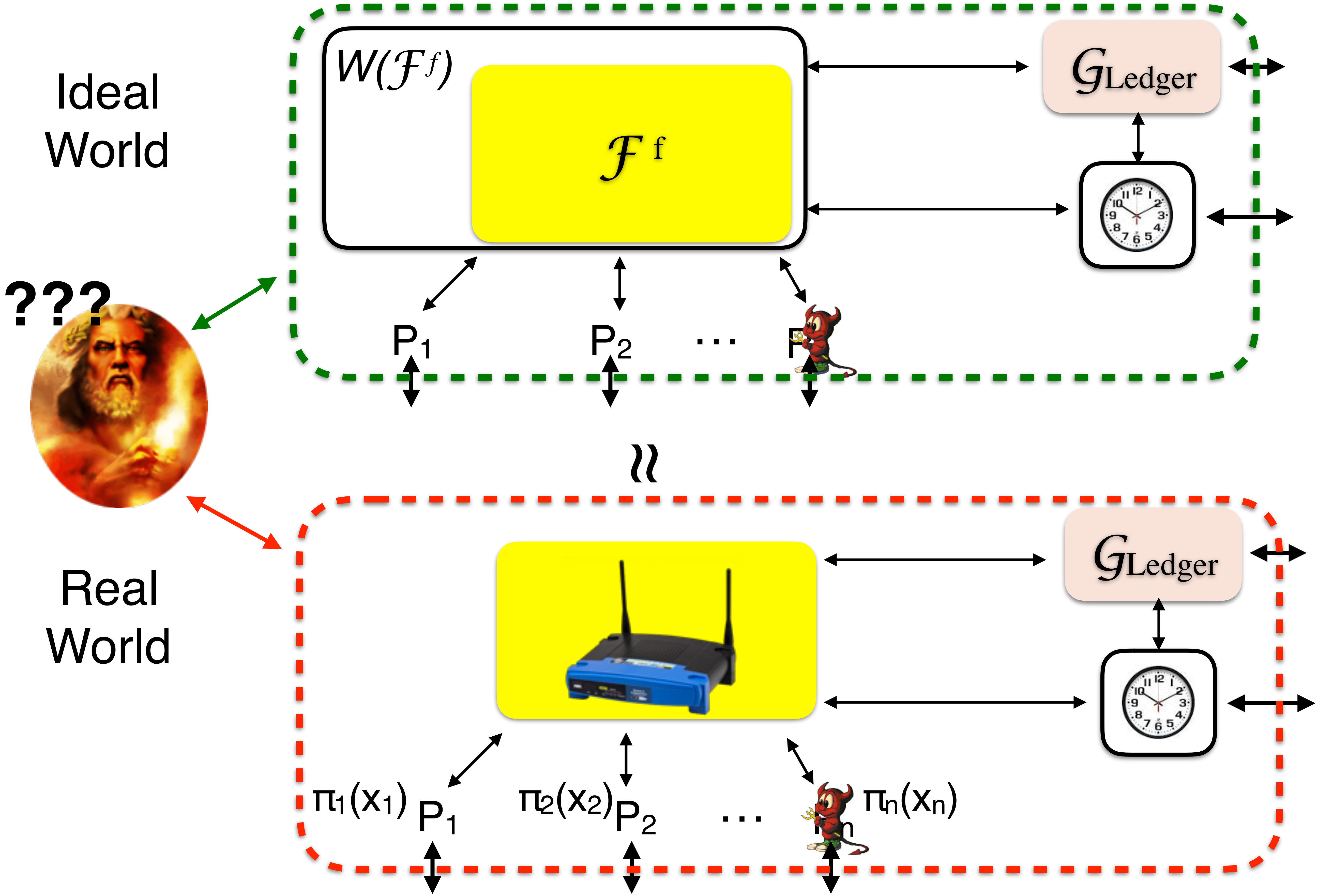
- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$

The adversary can deliver to P_i only if $Q^{Dlvr}(State, P_i)$=True

The adversary can make P_i abort only if $Q^{Abrt}(State, P_i)$=True



A Formal Model: GUC



Take Away Message and Open Directions

Take Away Message and Open Directions

- Bitcoin opens new directions for cryptographic models
  - Adding a reward/punishment mechanism restricts the set of likely attacks
  - Limitations of crypto should be reconsidered (Impossibilities/Efficiencies)
- The choice of the model makes a difference when suggesting a solution
  - Safe strategy: Rectify the cryptographic model (Bonus: compatibility)

Take Away Message and Open Directions

- Bitcoin opens new directions for cryptographic models
  - Adding a reward/punishment mechanism restricts the set of likely attacks
  - Limitations of crypto should be reconsidered (Impossibilities/Efficiencies)
- The choice of the model makes a difference when suggesting a solution
  - Safe strategy: Rectify the cryptographic model (Bonus: compatibility)

Future directions

- A game theoretic analysis might allow us to improve existing results
- What more can we get from Bitcoin?
- The right model for exploring its rational aspects?