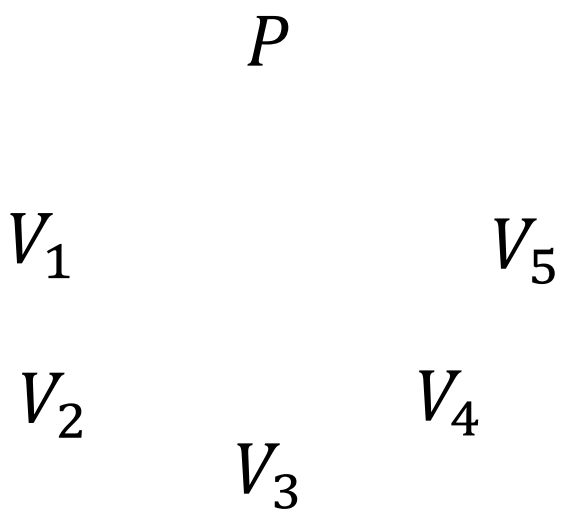
Your Reputation's Safe with Me: Framing-Free Distributed Zero-Knowledge Proofs

Carmit Hazay — Bar-Ilan University

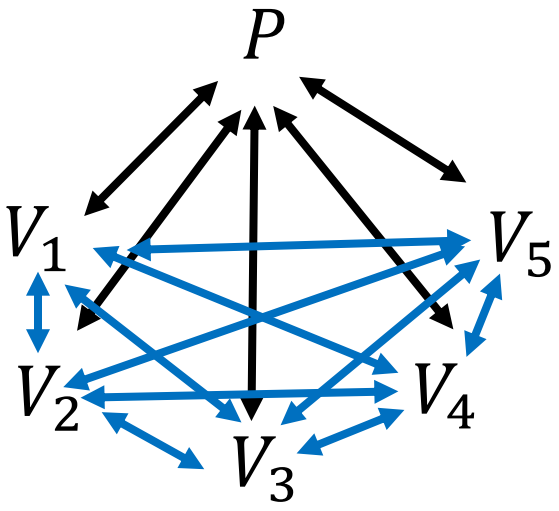
Muthuramakrishnan Venkitasubramaniam — Georgetown University

Mor Weiss — Bar-Ilan University

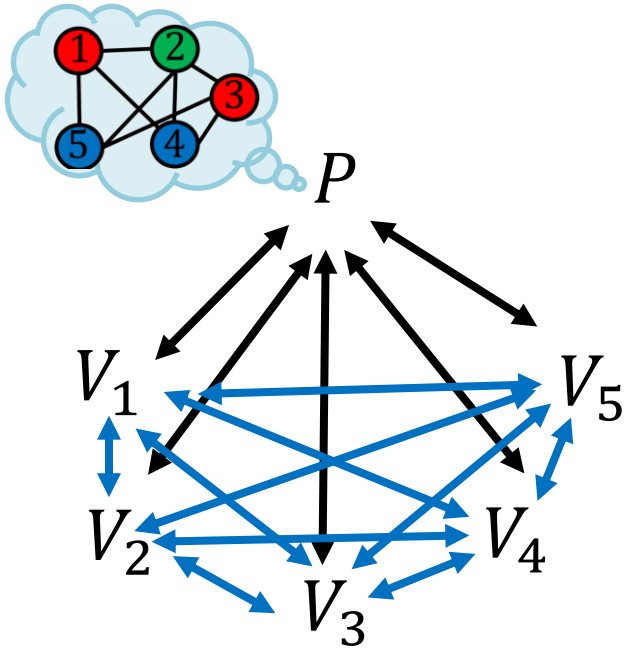
Framing-Free Distributed ZK Proofs



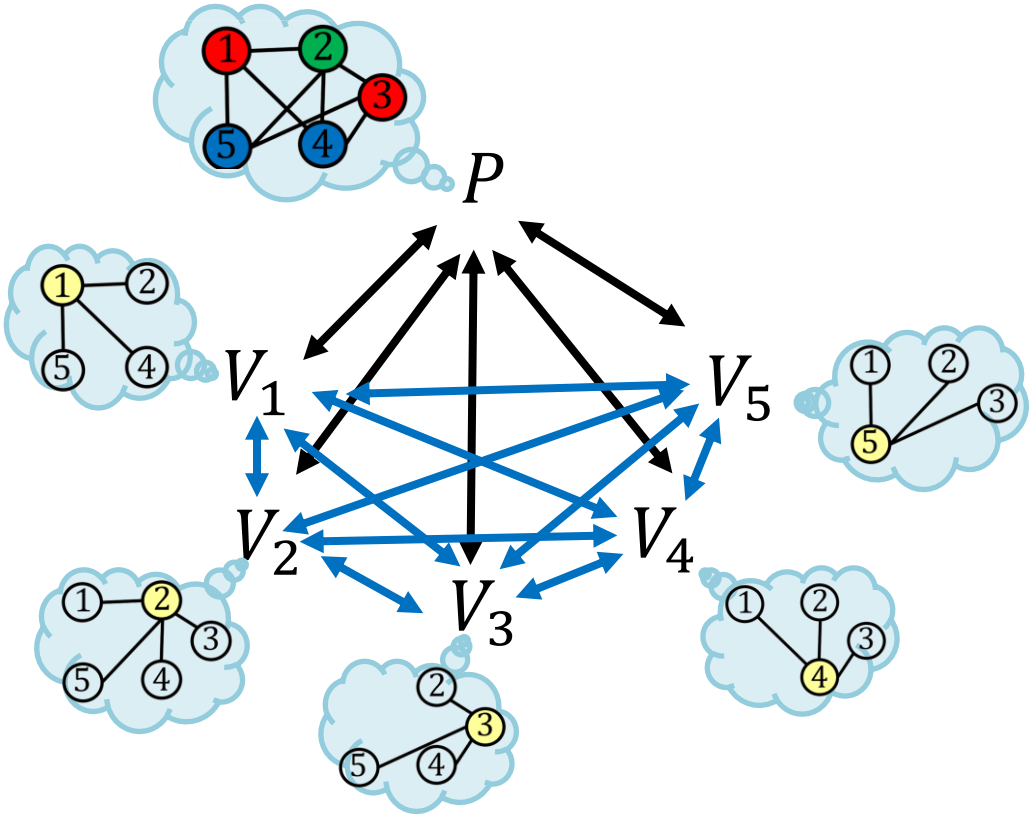
Framing-Free Distributed ZK Proofs



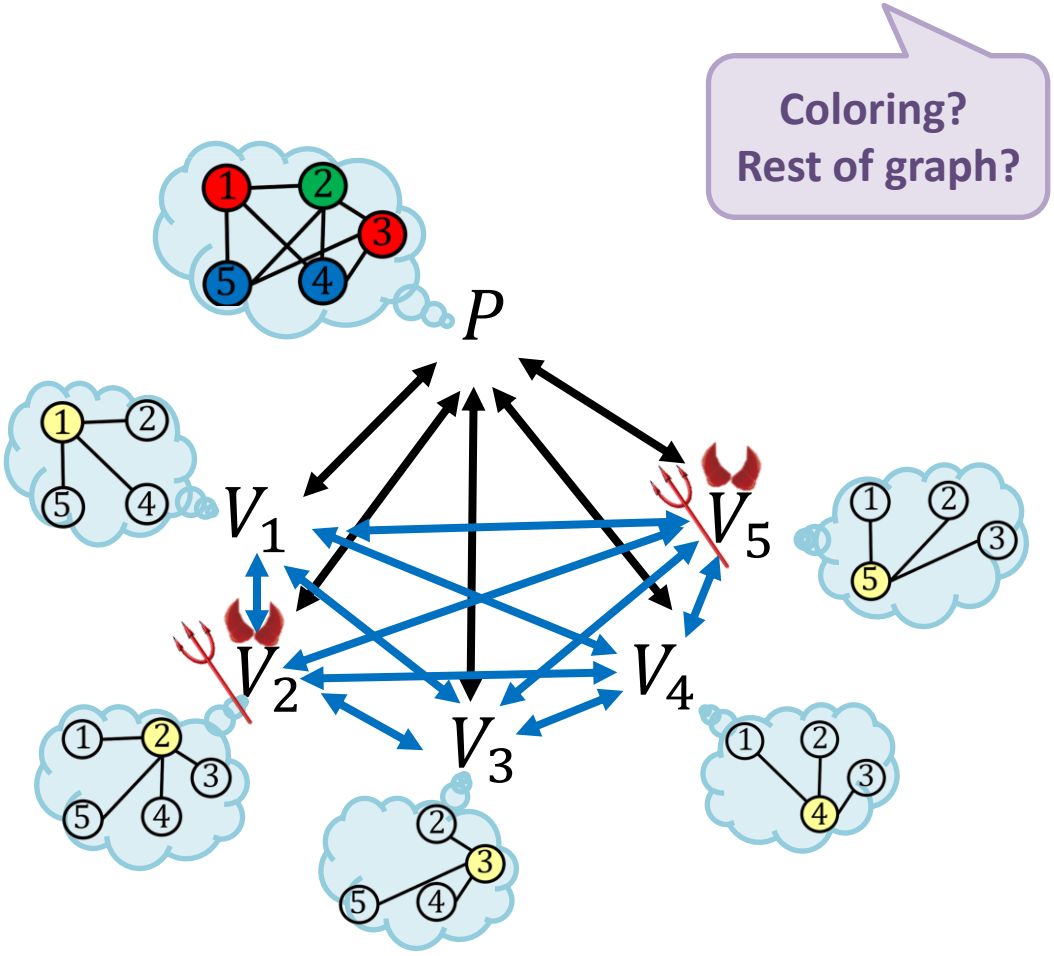
Framing-Free Distributed ZK Proofs



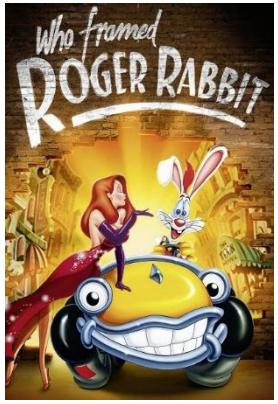
Framing-Free Distributed ZK Proofs

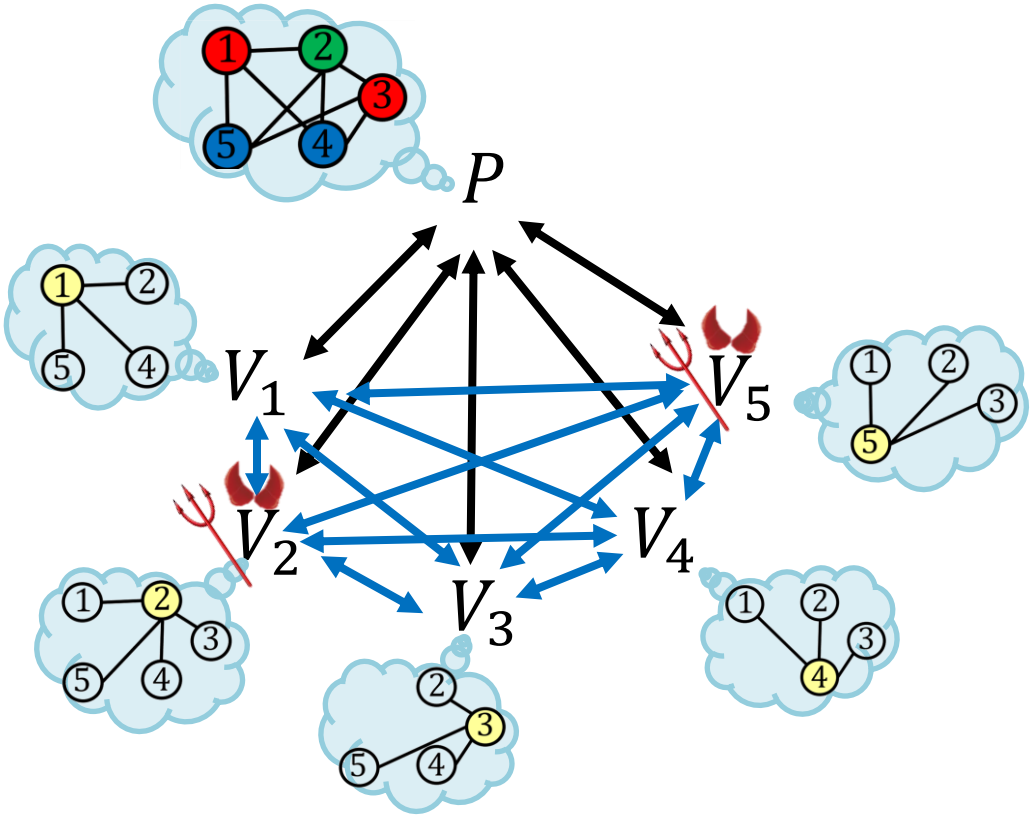


Framing-Free Distributed ZK Proofs



Framing-Free Distributed ZK Proofs



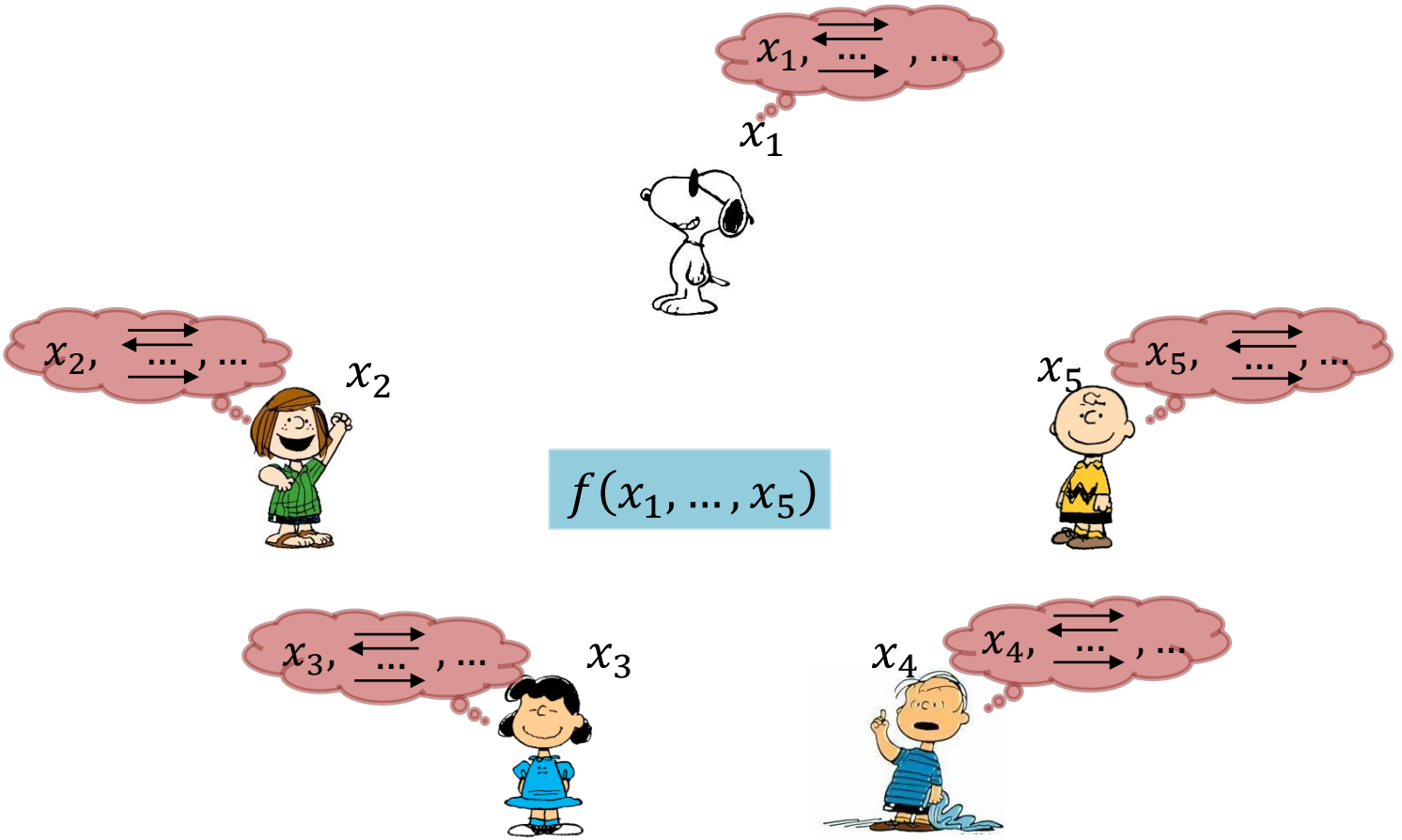


Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement

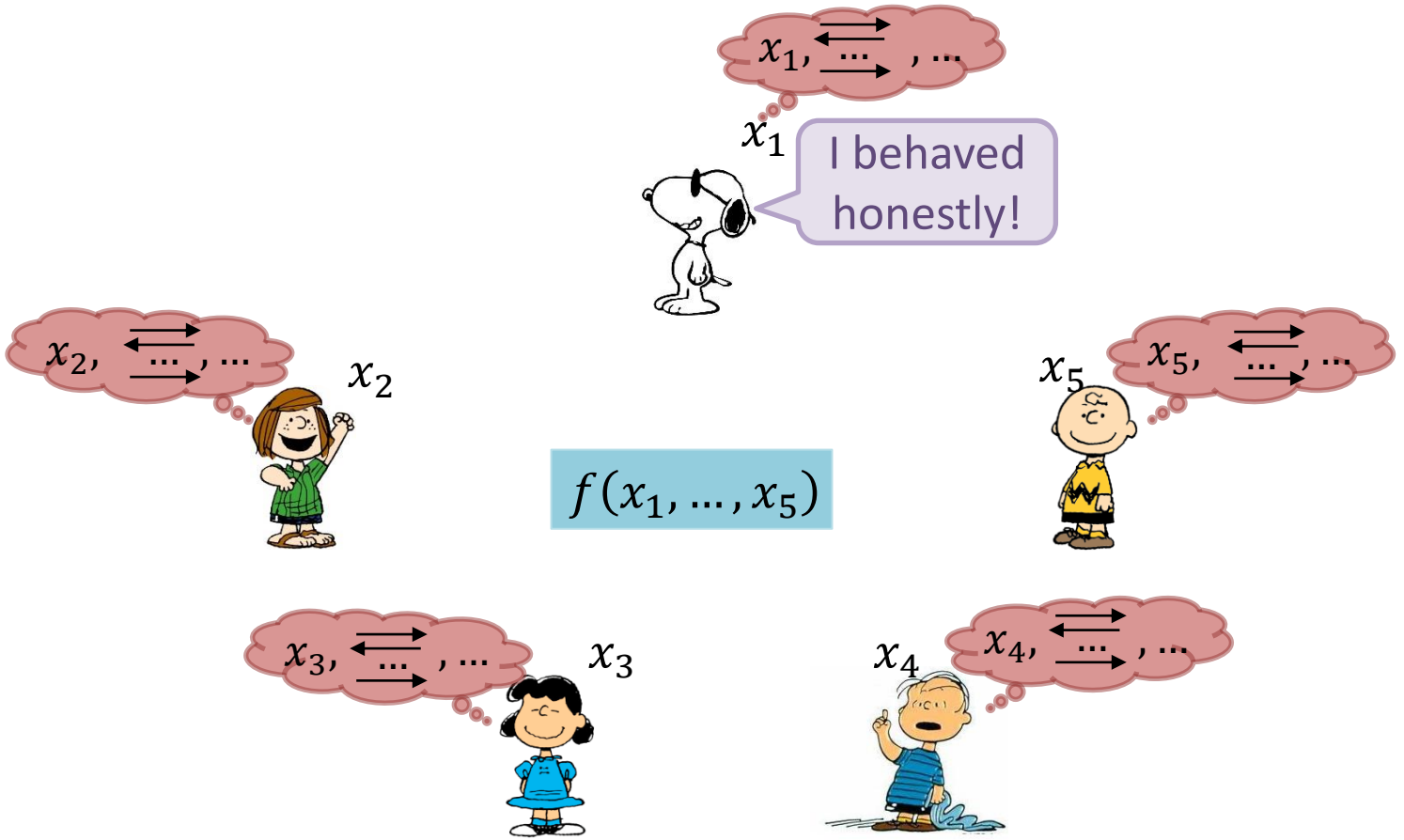
Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



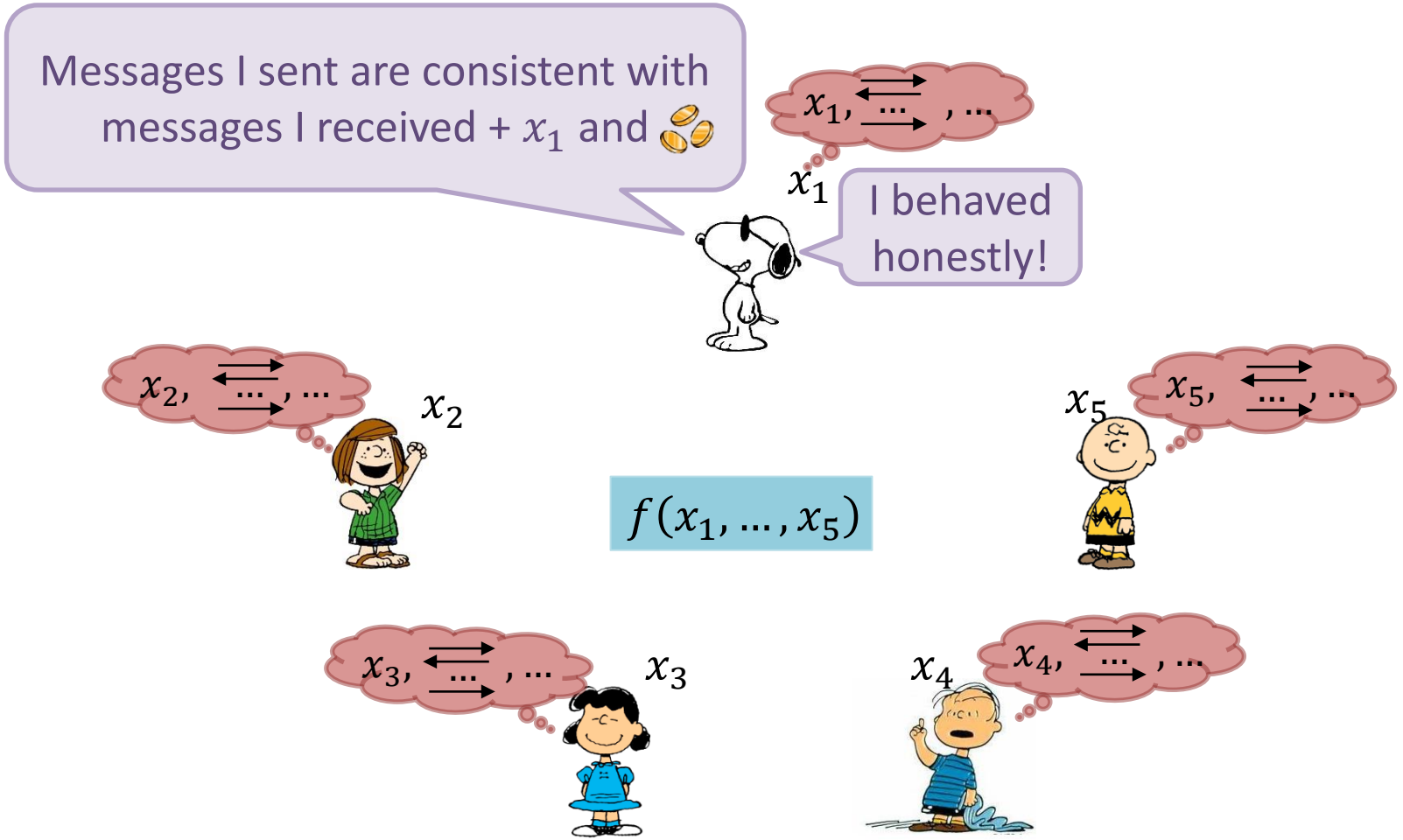
Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



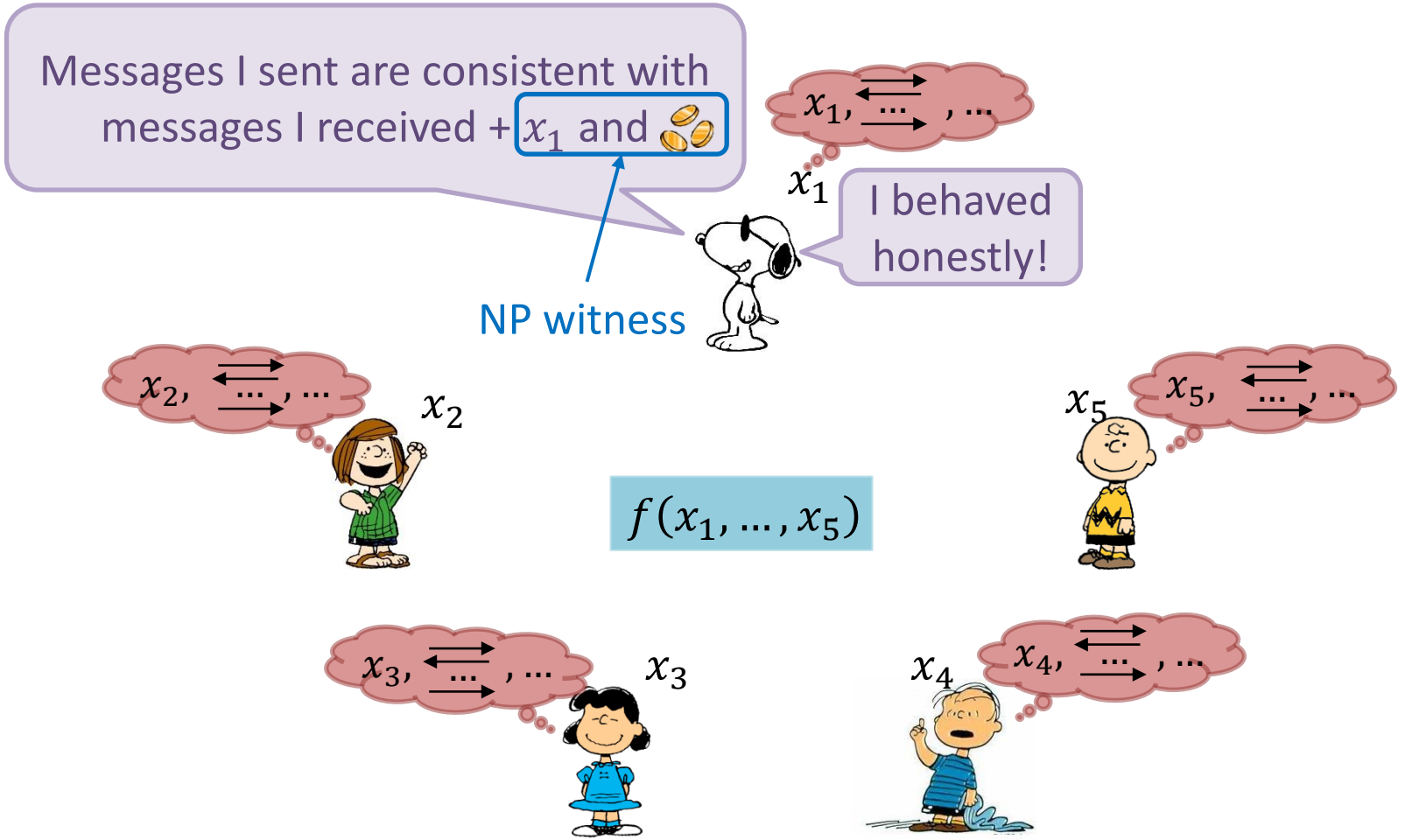
Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



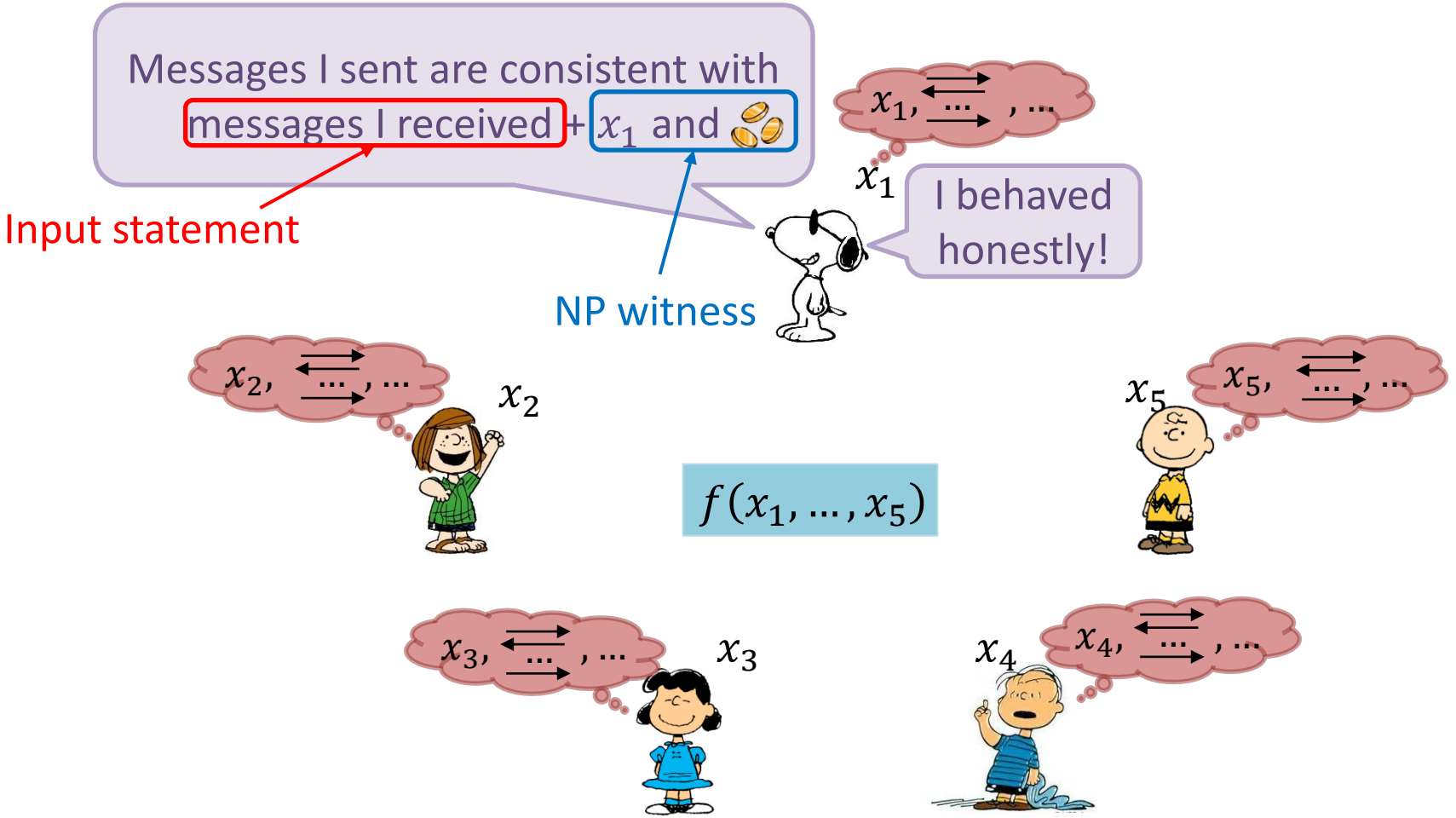
Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



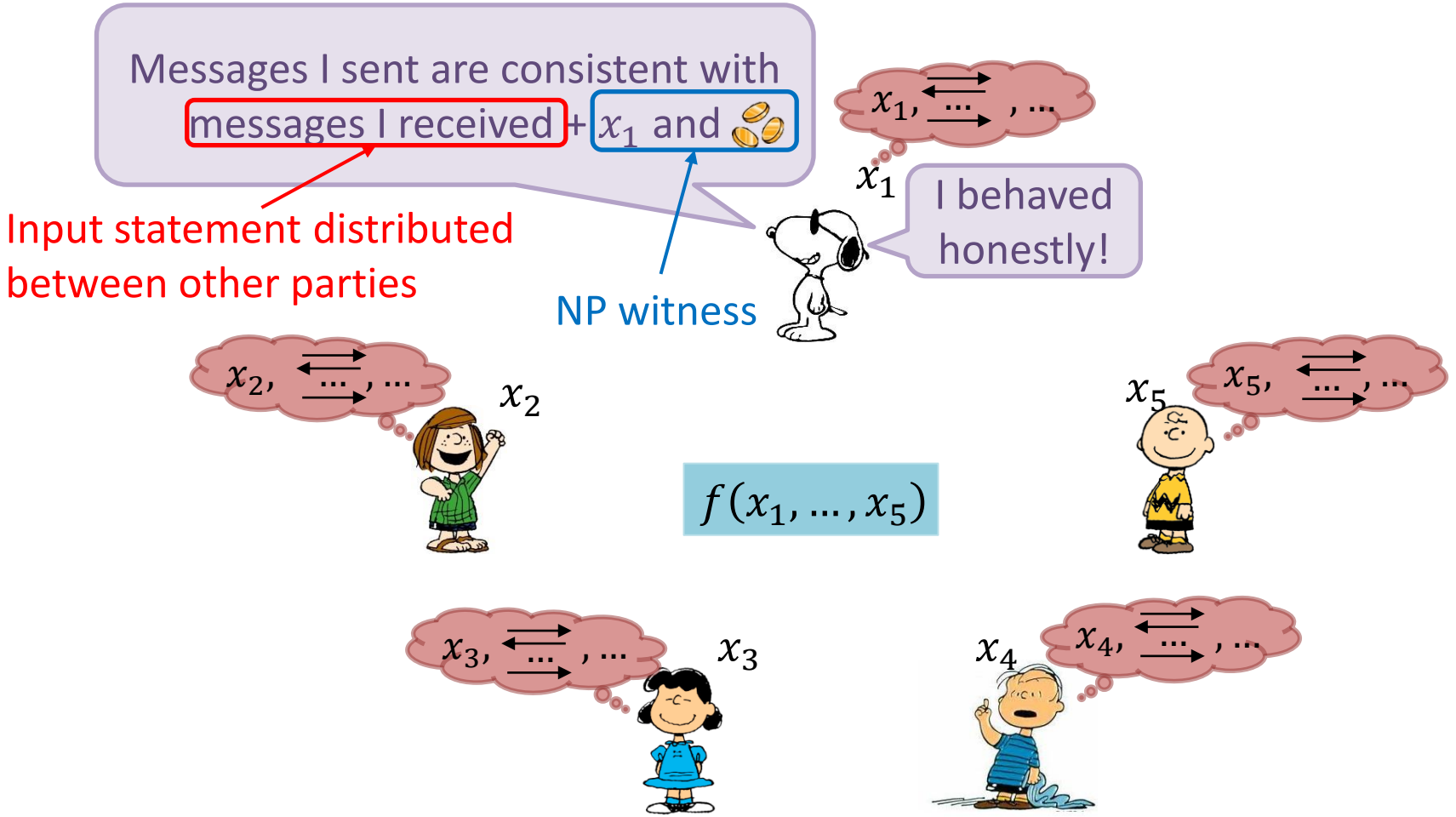
Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement



Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement
- Useful in many situations
  - Proving honest behavior in distributed protocols
  - Aggregate statistics (proofs on secret-shared data)
  - Verifiable secret sharing
  - …

Why **Distributed** ZK (dZK)?

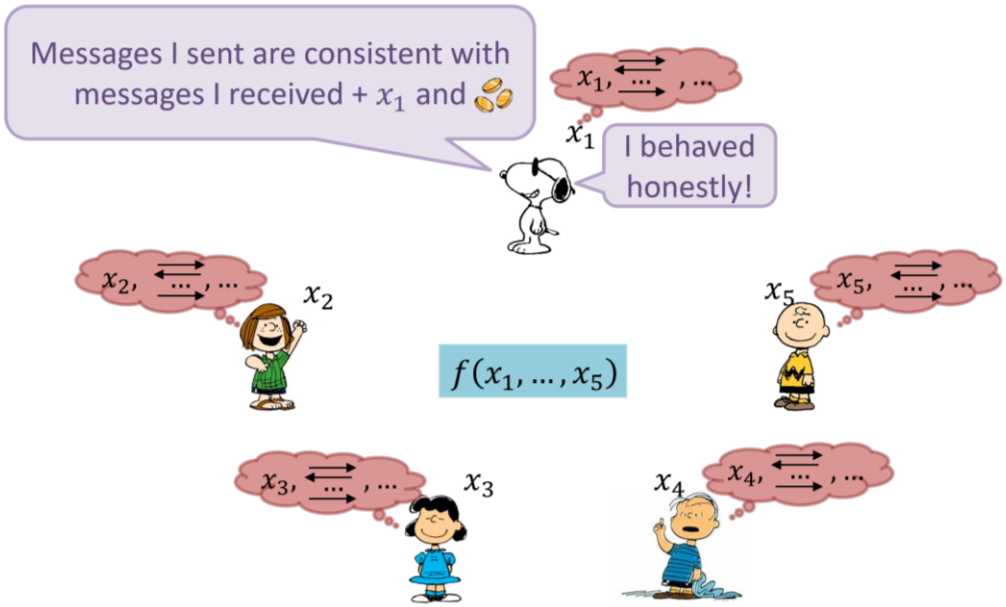
- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement
- Useful in many situations
  - Proving honest behavior in distributed protocols
  - Aggregate statistics (proofs on secret-shared data)
  - Verifiable secret sharing
  - …
- **Framing a real concern in such scenarios**

Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement
- Useful in many situations
  - Proving honest behavior in distributed protocols
  - Aggregate statistics (proofs on secret-shared data)
  - Verifiable secret sharing
  - …
- **Framing a real concern in such scenarios**
- **Verification efficiency** desirable, especially in 2-phase applications:
  - “Proving phase” (offline, can be executed in parallel to external application\execution)
  - “Verification phase” (online)

Why **Distributed** ZK (dZK)?

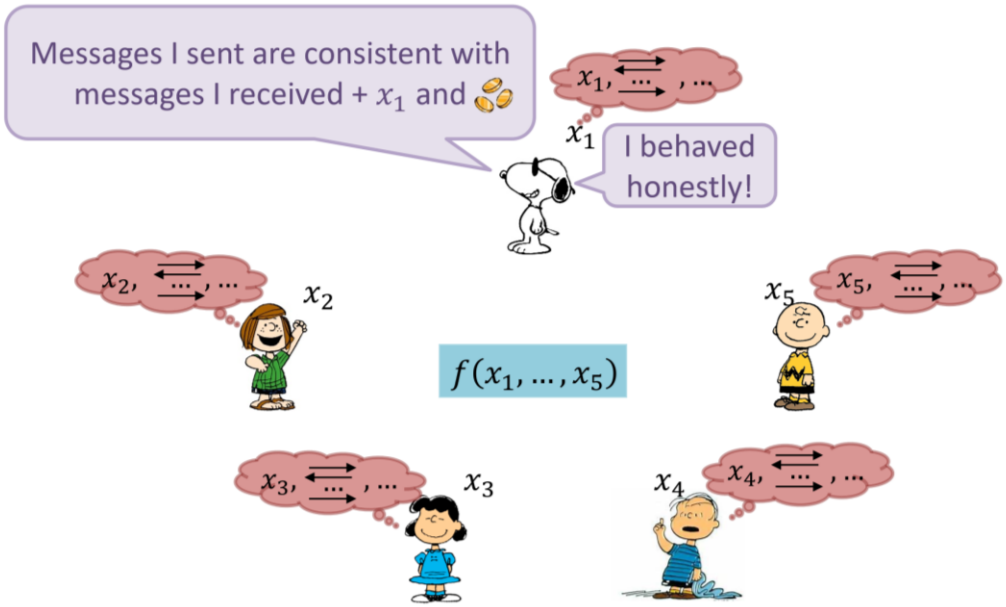
- Many scena erifiers
- Each verifie
- Useful in m
  - Proving h
  - Aggregate
  - Verifiable
  - …



- **Framing a real concern in such scenarios**
- **Verification efficiency** desirable, especially in 2-phase applications:
  - “Proving phase” (offline, can be executed in parallel to external application\execution)
  - “Verification phase” (online)

Why **Distributed** ZK (dZK)?

- Many scena erifiers
- Each verifie
- Useful in m
  - Proving h
  - Aggregate
  - Verifiable
  - …



- **Framing a real concern in such scenarios**
- **Verification efficiency** desirable, especially in 2-phase applications:

  $\boldsymbol{poly(k, \log|x|)}$ **Total CC during verification phase**

  - "Proving phase" (offline, can be executed in parallel to external application\execution)
  - "Verification phase" (online)

Why **Distributed** ZK (dZK)?

- Many scenarios involve single prover and many verifiers
- Each verifier has only local view of statement
- Useful in many situations
  - Proving honest behavior in distributed protocols
  - Aggregate statistics (proofs on secret-shared data)
  - Verifiable secret sharing
  - …
- **Framing a real concern in such scenarios**
- **Verification efficiency** desirable, especially in 2-phase applications:
  - "Proving phase" (offline, can be executed in parallel to external application\execution)
  - "Verification phase" (online)

$\boldsymbol{poly}(\boldsymbol{k}, \log|\boldsymbol{x}|)$ **Total CC during verification phase**

- **Our goal:** verification-efficient framing free dZKs

Distributed ZK Proofs: Previous Works

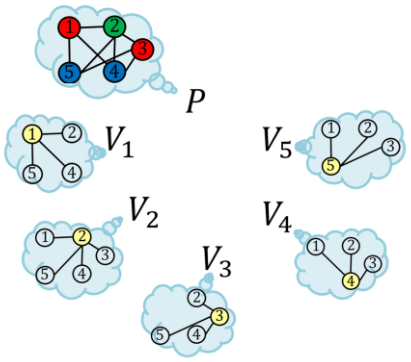
- **Many different models**
  - Verifiers colluding with prover, computational powers of corrupted parties, who knows input statement, … [ABD91,CF02,GIKR02,GO07,CBM15,CB17, BBCGI19,AKP20a,BBGIN20,BGIN21,BJOSS22,WY22, AKP22]

Distributed ZK Proofs: Previous Works

- **Many different models**
  - Verifiers colluding with prover, computational powers of corrupted parties, who knows input statement, … [ABD91,CF02,GIKR02,GO07,CBM15,CB17, BBCGI19,AKP20a,BBGIN20,BGIN21,BJOSS22,WY22, AKP22]
  - Related notions (e.g, VRS [GIKR02,AKP20a,AKP22])

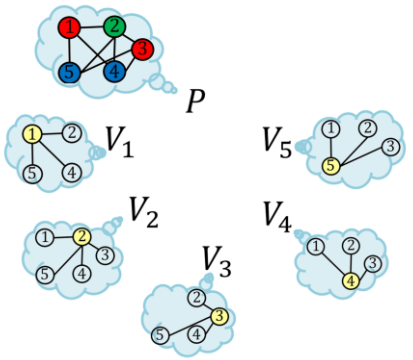
Distributed ZK Proofs: Previous Works

- **Many different models**
  - Verifiers colluding with prover, computational powers of corrupted parties, who knows input statement, … [ABD91,CF02,GIKR02,GO07,CBM15,CB17, BBCGI19,AKP20a,BBGIN20,BGIN21,BJOSS22,WY22, AKP22]
  - Related notions (e.g, VRS [GIKR02,AKP20a,AKP22])
  - Most related model [BBCGI19]
    - Input statement x distributed between verifiers
    - Soundness against prover colluding with verifiers
    - Information-theoretic security



Distributed ZK Proofs: Previous Works

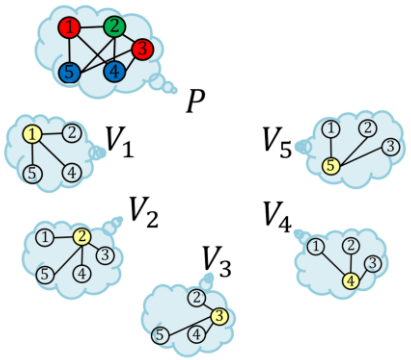
- **Many different models**
  - Verifiers colluding with prover, computational powers of corrupted parties, who knows input statement, … [ABD91,CF02,GIKR02,GO07,CBM15,CB17, BBCGI19,AKP20a,BBGIN20,BGIN21,BJOSS22,WY22, AKP22]
  - Related notions (e.g, VRS [GIKR02,AKP20a,AKP22])
  - Most related model [BBCGI19]
    - Input statement x distributed between verifiers
    - Soundness against prover colluding with verifiers
    - Information-theoretic security



  - Constructions [ACF02,GIKR02,GO14,BBCGI19,AKP20a,BJOSS22,WY22,AKP22]
    - Verification-efficient dZK for NP, but prover can be framed [BBCGI19]

Distributed ZK Proofs: Previous Works

- **Many different models**
  - Verifiers colluding with prover, computational powers of corrupted parties, who knows input statement, … [ABD91,CF02,GIKR02,GO07,CBM15,CB17, BBCGI19,AKP20a,BBGIN20,BGIN21,BJOSS22,WY22, AKP22]
  - Related notions (e.g, VRS [GIKR02,AKP20a,AKP22])
  - Most related model [BBCGI19]
    - Input statement x distributed between verifiers
    - Soundness against prover colluding with verifiers
    - Information-theoretic security



  - Constructions [ACF02,GIKR02,GO14,BBCGI19,AKP20a,BJOSS22,WY22,AKP22]
    - Verification-efficient dZK for NP, but prover can be framed [BBCGI19]
- **Generic MPC protocols [BGW88,CCD88] give dZK**
  - Even with $O(1)$ rounds [IK02,ABT19,ACGJ18,ACGJ19] and tight thresholds and round complexities [AKP20a,AKP20b]
  - **Not verification-efficient!**

Our Contribution

- **First verification-efficient "framing-free" dZK for NP**
  - k verifiers, $t < \frac{k-2}{6}$ corruptions
  - Or $t < \frac{k}{2}$ without framing-free (matches dZKs of [BBCGI19])

Our Contribution

- **First verification-efficient "framing-free" dZK for NP**
  - k verifiers, $t < \frac{k-2}{6}$ corruptions
  - Or $t < \frac{k}{2}$ without framing-free (matches dZKs of [BBCGI19])
- **New approach to dZK design** (even without framing-free)
  - Based on "MPC in the head"
    - Using new analysis for "MPC in the head" in distributed setting (fundamentally different from the analysis in [AKP22])
  - Constructions of [BBCGI19] based on fully-linear IOPs

Our Contribution

- **First verification-efficient "framing-free" dZK for NP**
  - k verifiers, $t < \frac{k-2}{6}$ corruptions
  - Or $t < \frac{k}{2}$ without framing-free (matches dZKs of [BBCGI19])
- **New approach to dZK design** (even without framing-free)
  - Based on "MPC in the head"
    - Using new analysis for "MPC in the head" in distributed setting (fundamentally different from the analysis in [AKP22])
  - Constructions of [BBCGI19] based on fully-linear IOPs
- **Instantiations** (assuming ideal coin tossing, $\Omega(k)$ corruptions):

# Rounds	Total proof length	Verification CC
3	$O(\log k \cdot \log\|C\| \cdot \|C\|)$	$O(k^2)$
4	$O(\|C\|)$	$O(k^2 + s)$ (s statistical security param)

Our Contribution

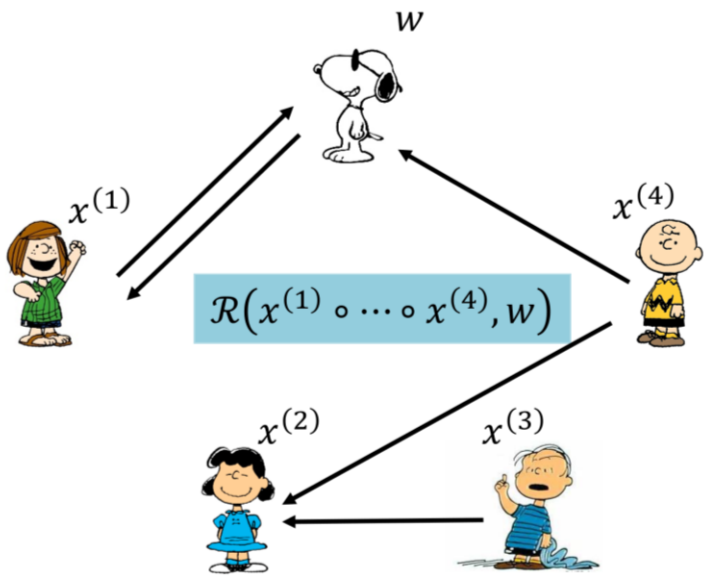
- **First verification-efficient "framing-free" dZK for NP**
  - k verifiers, $t < \frac{k-2}{6}$ corruptions
  - Or $t < \frac{k}{2}$ without framing-free (matches dZKs of [BBCGI19])
- **New approach to dZK design** (even without framing-free)
  - Based on "MPC in the head"
    - Using new analysis for "MPC in the head" in distributed setting (fundamentally different from the analysis in [AKP22])
  - Constructions of [BBCGI19] based on fully-linear IOPs
- **Instantiations** (assuming ideal coin tossing, $\Omega(k)$ corruptions):

# Rounds	Total proof length	Verification CC
3	$O(\log k \cdot \log\|C\| \cdot \|C\|)$	$O(k^2)$
4	$O(\|C\|)$	$O(k^2 + s)$ (s statistical security param)

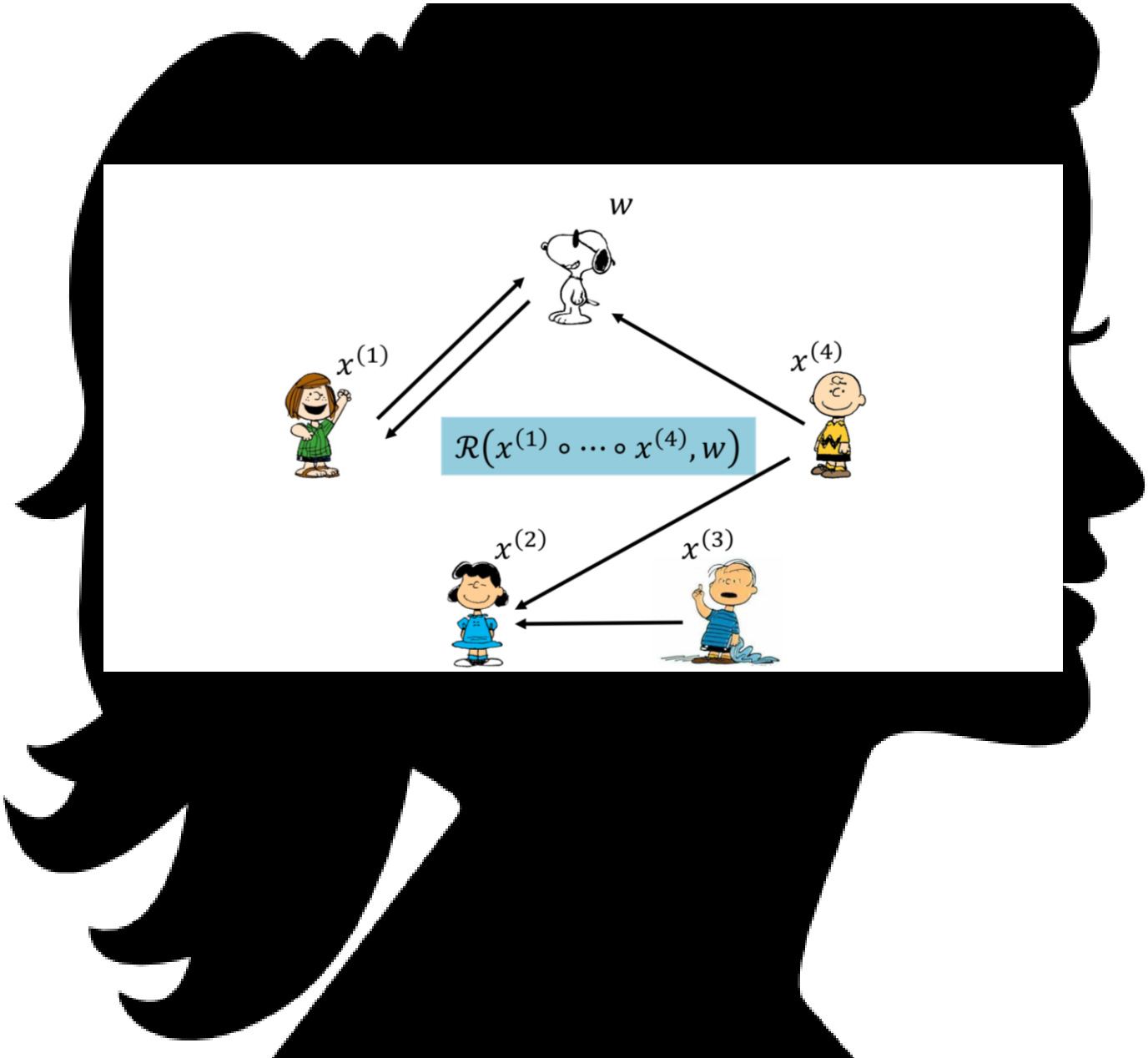
- **Applications:** aggregate statistics, VSS, (reusable) certifiable VSS, proving honest behavior

Highlights of Our dZK Construction

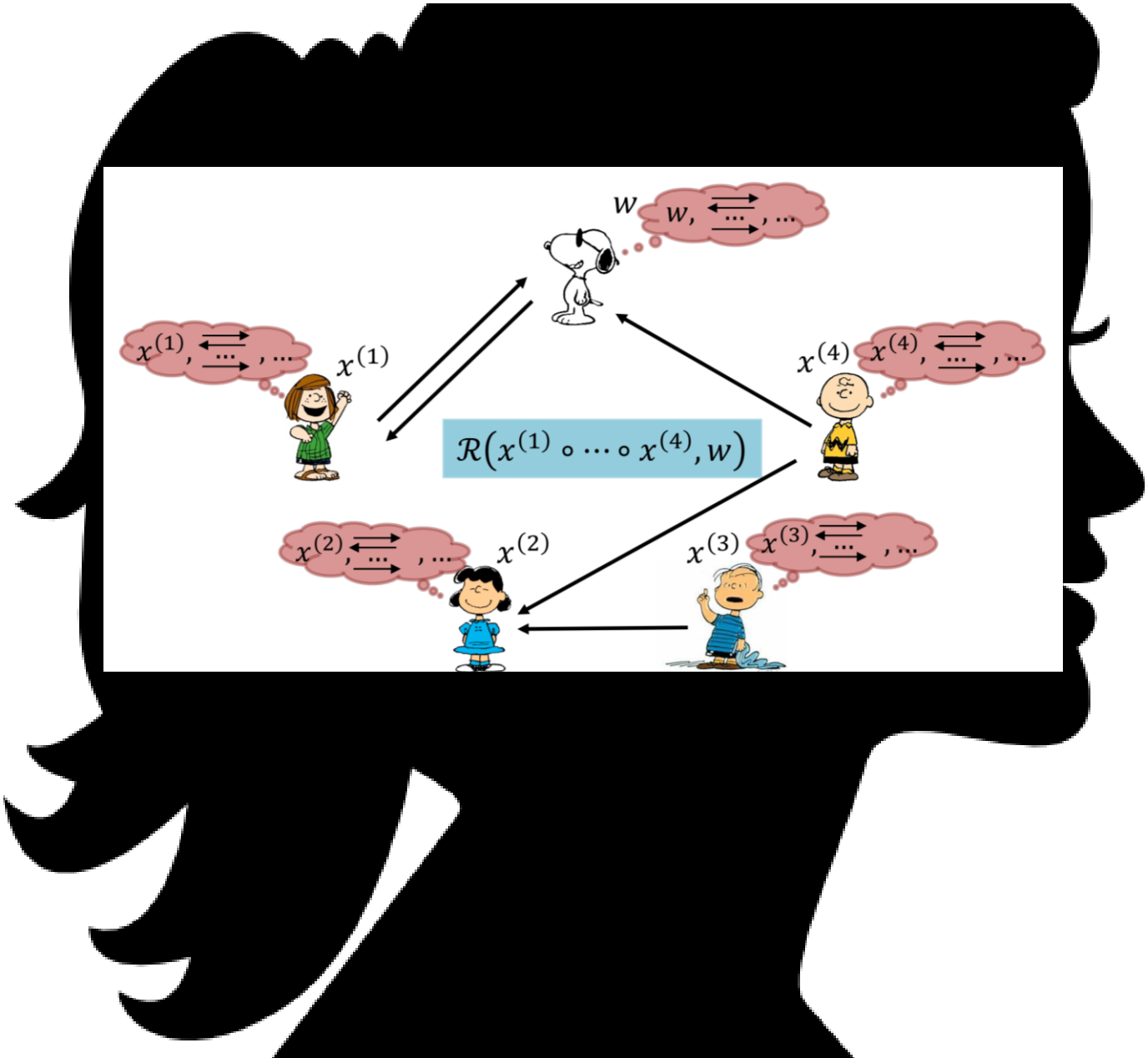
Proofs from “MPC in the Head”



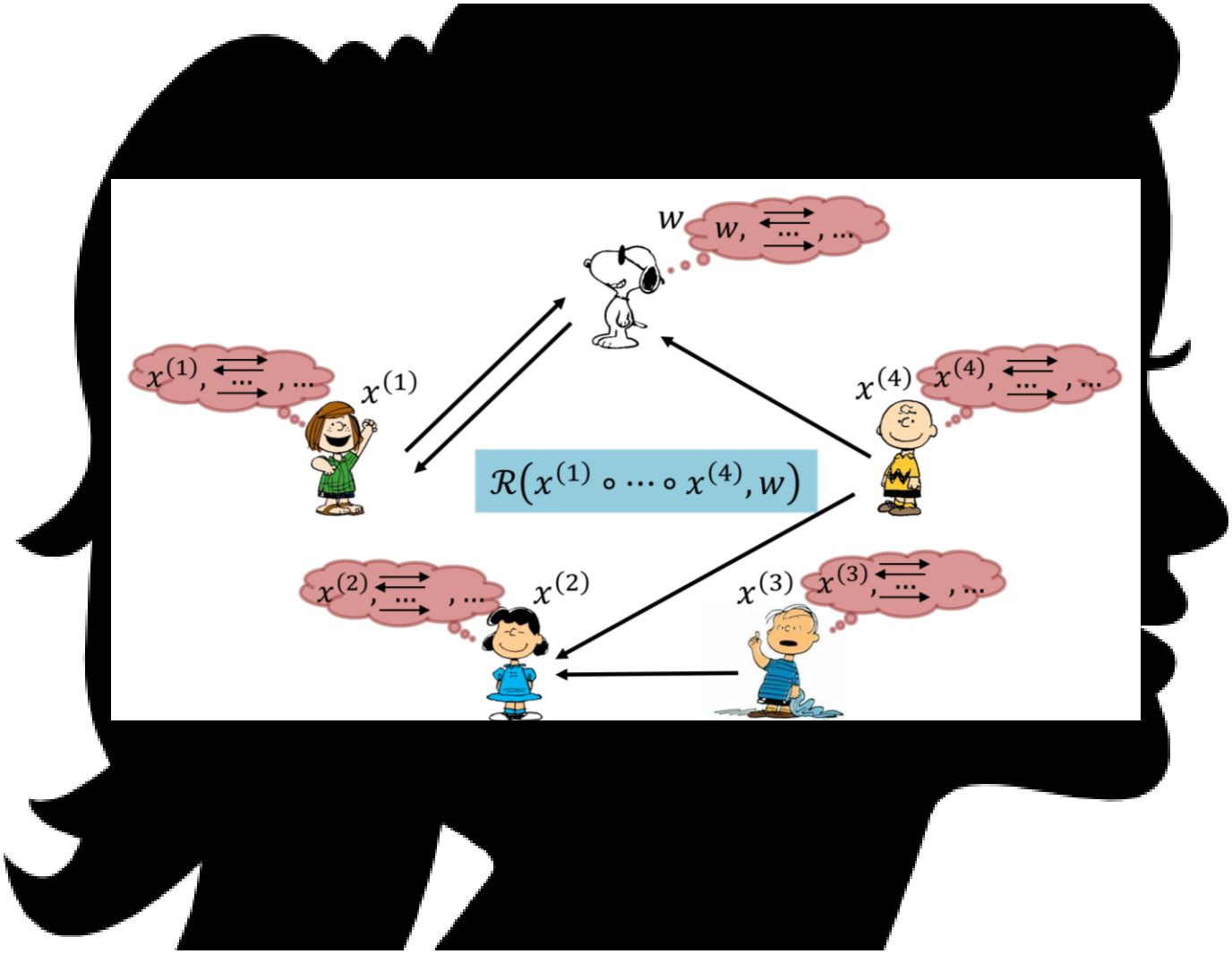
Proofs from “MPC in the Head”



Proofs from “MPC in the Head”



Proofs from “MPC in the Head”



Pairwise consistency ⇒ global consistency ⇒ soundness [IKOS07]

dZK from “MPC in the Head” (Warmup)

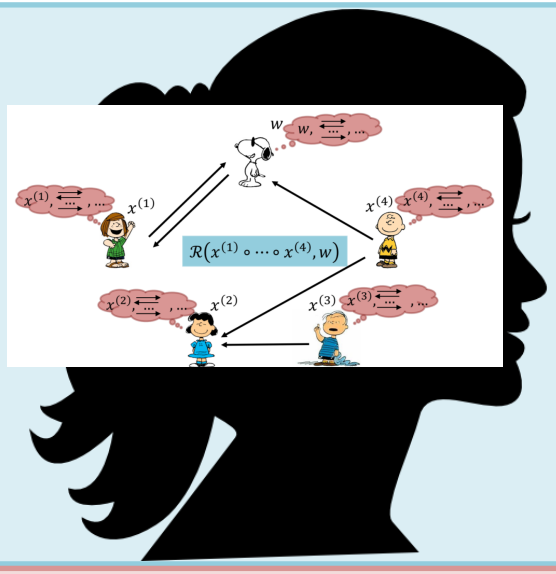
Prover Zone

$V_1^{x^{(1)}}$ $V_2^{x^{(2)}}$ $V_3^{x^{(3)}}$ $V_4^{x^{(4)}}$

Verification Zone

dZK from “MPC in the Head” (Warmup)

Prover Zone

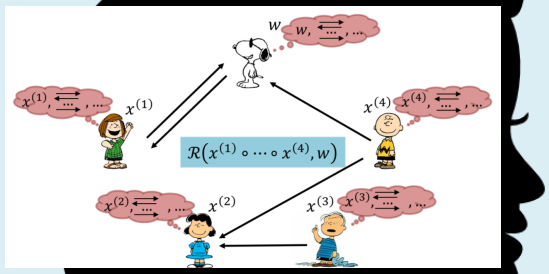


$V_1^{x^{(1)}}$ $V_2^{x^{(2)}}$ $V_3^{x^{(3)}}$ $V_4^{x^{(4)}}$

Verification Zone

dZK from “MPC in the Head” (Warmup)

Prover Zone



$x^{(1)}$

$x^{(4)}$

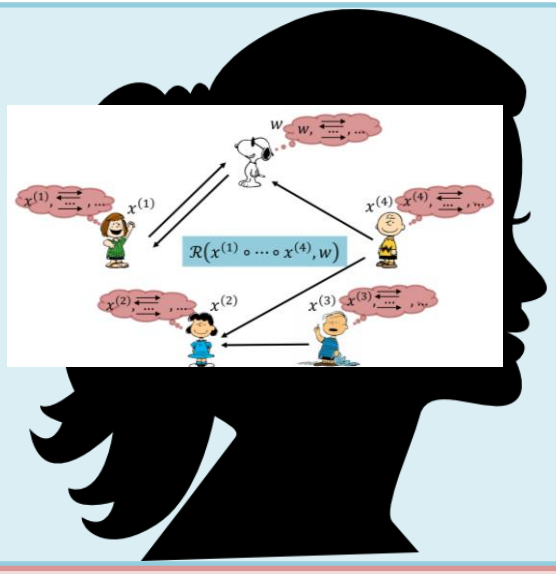
...

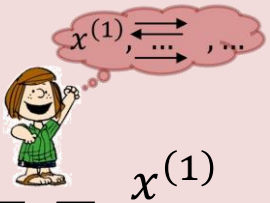
V_1 $x^{(1)}$ V_2 $x^{(2)}$ V_3 $x^{(3)}$ V_4 $x^{(4)}$

Verification Zone

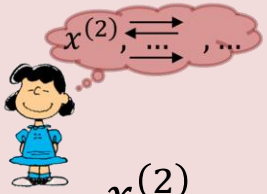
dZK from “MPC in the Head” (Warmup)

Prover Zone

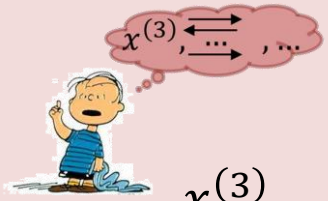




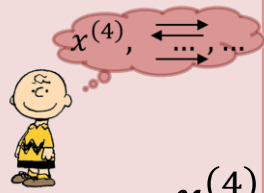
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$

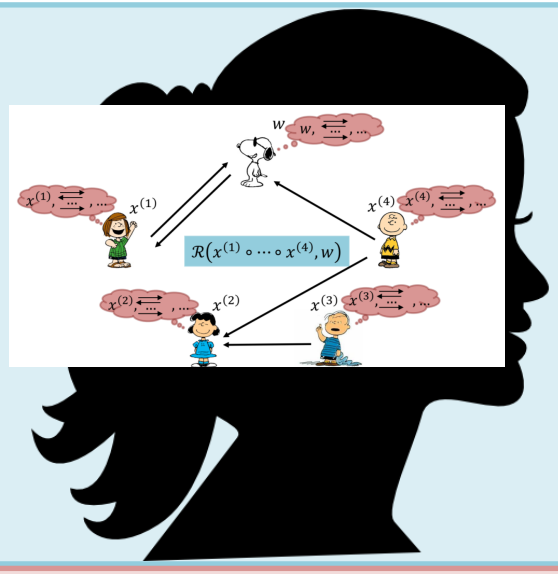


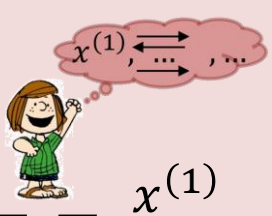
V_4 $x^{(4)}$

Verification Zone

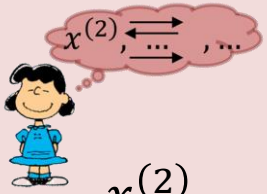
dZK from “MPC in the Head” (Warmup)

Prover Zone

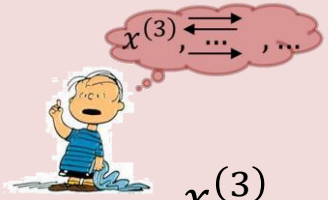




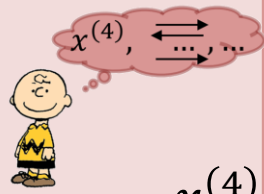
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

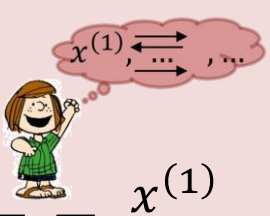
Each V_i checks:

- Local consistency: input $x^{(i)}$ and output 1
- Pairwise consistency with every V_j

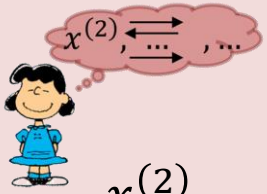
Accept if no verifier b\casts complaint

Verification Zone

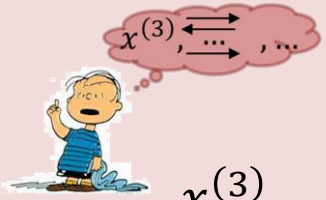
Getting Verification Efficiency



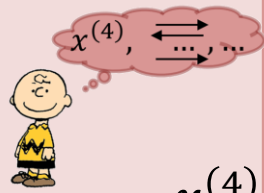
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

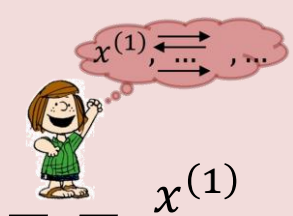
Each V_i checks:

- Local consistency: input $x^{(i)}$ and output 1
- Pairwise consistency with every V_j

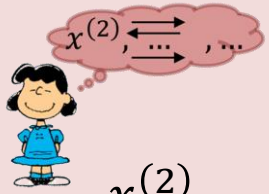
Accept if no verifier b\casts complaint

Getting Verification Efficiency

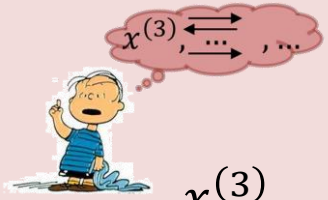
- **To get verification efficiency:** compress communication with information-theoretic MACs
  - Verifiers exchange only tags
  - All MACs generated using one random coin (from the oracle)



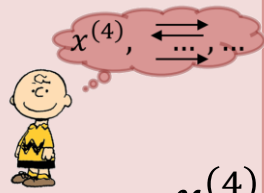
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

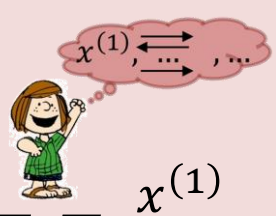
Each V_i checks:

- Local consistency: input $x^{(i)}$ and output 1
- Pairwise consistency with every V_j

Accept if no verifier b\casts complaint

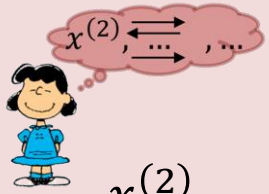
Getting Verification Efficiency

- **To get verification efficiency:** compress communication with information-theoretic MACs
  - Verifiers exchange only tags
  - All MACs generated using **one** random coin (from the oracle)

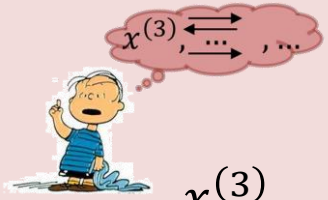


V_1 $x^{(1)}$

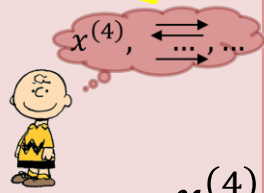
$t_{1,2} =^{?} t_{2,1}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

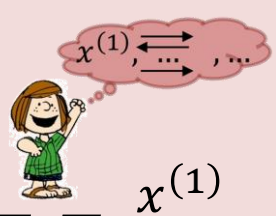
Each V_i checks:

- Local consistency
- Pairwise consistency with every V_j **using tags**

Accept if no verifier b\casts complaint

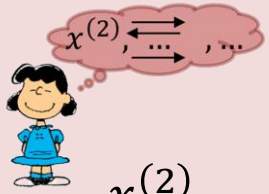
Getting Verification Efficiency

- **To get verification efficiency:** compress communication with information-theoretic MACs
  - Verifiers exchange only tags
  - All MACs generated using **one** random coin (from the oracle)
- Matches dZK of [BBCGI19]

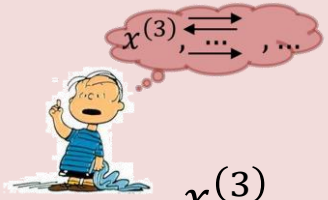


V_1 $x^{(1)}$

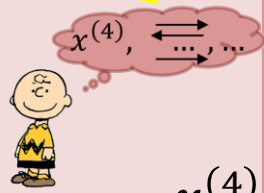
$t_{1,2} =^{?} t_{2,1}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



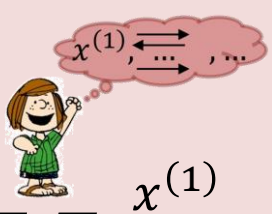
V_4 $x^{(4)}$

Each V_i checks:

- Local consistency
- Pairwise consistency with every V_j **using tags**

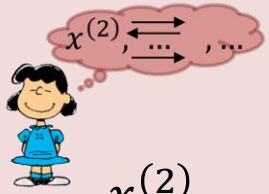
Accept if no verifier b\casts complaint

Getting Framing Free

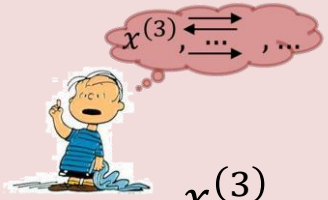


V_1 $x^{(1)}$

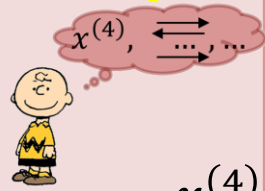
$t_{1,2} =^{?} t_{2,1}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

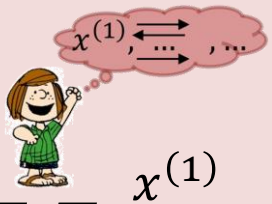
Each V_i checks:

- Local consistency
- Pairwise consistency with every V_j **using tags**

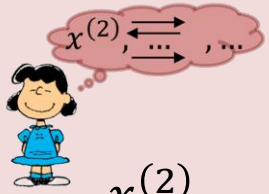
Accept if no verifier b\casts complaint

Getting Framing Free

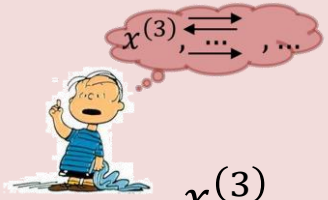
- **To get framing free:** All values b\casted, no P2P communication



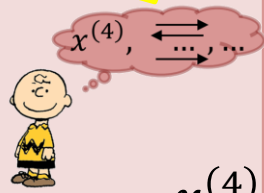
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

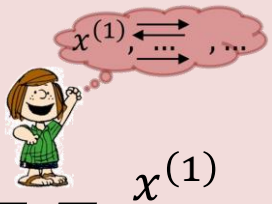
Each V_i checks:

- Local consistency
- Pairwise consistency with every V_j **using tags**

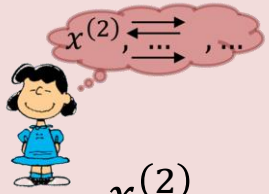
Accept if no verifier b\casts complaint

Getting Framing Free

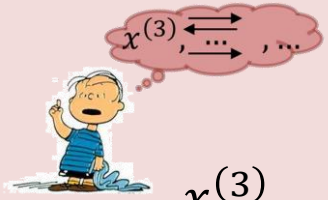
- **To get framing free:** All values b\casted, no P2P communication



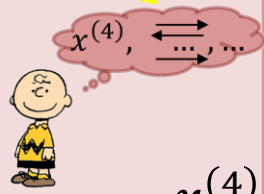
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

Each V_i checks:

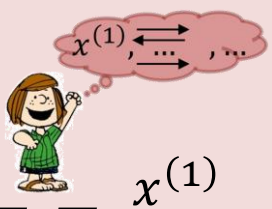
- Local consistency
- Pairwise consistency with every V_j **using tags**

Accept if no verifier b\casts complaint

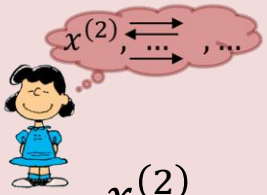
$t_{1,2}, t_{1,3}, t_{1,4}$
$view_4$ consistent?

Getting Framing Free

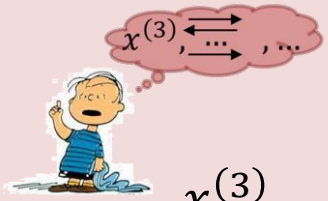
- **To get framing free:** All values b\casted, no P2P communication



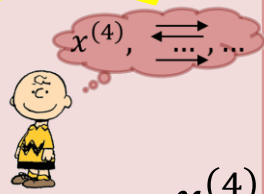
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



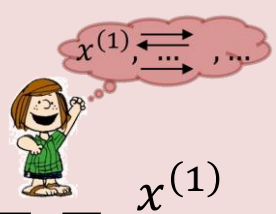
V_4 $x^{(4)}$

$t_{1,2}, t_{1,3}, t_{1,4}$
$view_4$ consistent?

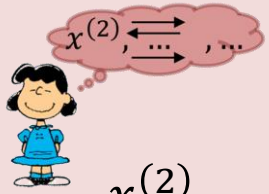
- C_1 := set of verifiers claiming local inconsistency
- C_2 := set of verifiers b\casting incorrect messages (P computes and b\casts)
- V_i accepts if $|C_1 \cup C_2| \leq t$, otherwise rejects

Getting Framing Free

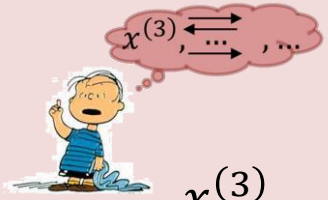
- **To get framing free:** All values b\casted, no P2P communication
- This is not ZK!



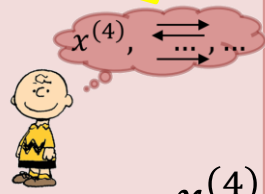
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



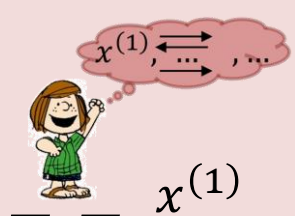
V_4 $x^{(4)}$

$t_{1,2}, t_{1,3}, t_{1,4}$
$view_4$ consistent?

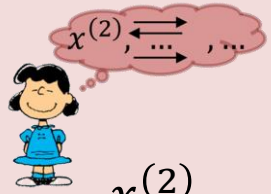
- C_1 := set of verifiers claiming local inconsistency
- C_2 := set of verifiers b\casting incorrect messages (P computes and b\casts)
- V_i accepts if $|C_1 \cup C_2| \leq t$, otherwise rejects

Getting Framing Free

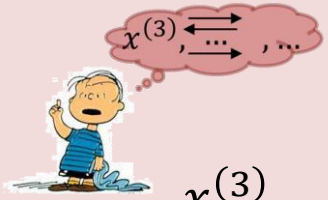
- **To get framing free:** All values b\casted, no P2P communication
- This is not ZK!
- **To preserve ZK:** masks tags with unique random masks, provided by prover
  - Unique mask for every pair V_i, V_j
  - Preserves soundness: masks chosen before random coin



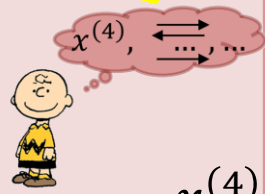
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



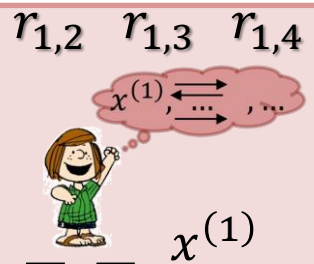
V_4 $x^{(4)}$

$t_{1,2}, t_{1,3}, t_{1,4}$
$view_4$ consistent?

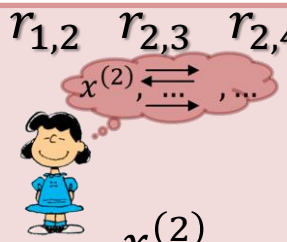
- C_1 := set of verifiers claiming local inconsistency
- C_2 := set of verifiers b\casting incorrect messages (P computes and b\casts)
- V_i accepts if $|C_1 \cup C_2| \leq t$, otherwise rejects

Getting Framing Free

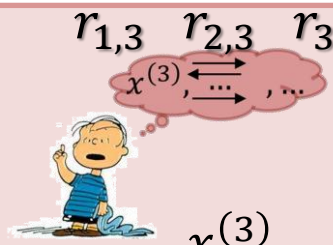
- **To get framing free:** All values b\casted, no P2P communication
- This is not ZK!
- **To preserve ZK:** masks tags with unique random masks, provided by prover
  - Unique mask for every pair V_i, V_j
  - Preserves soundness: masks chosen before random coin



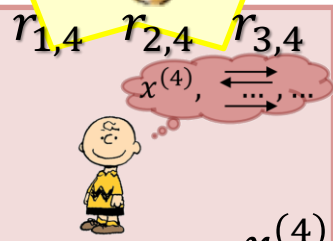
V_1 $x^{(1)}$



V_2 $x^{(2)}$



V_3 $x^{(3)}$



V_4 $x^{(4)}$

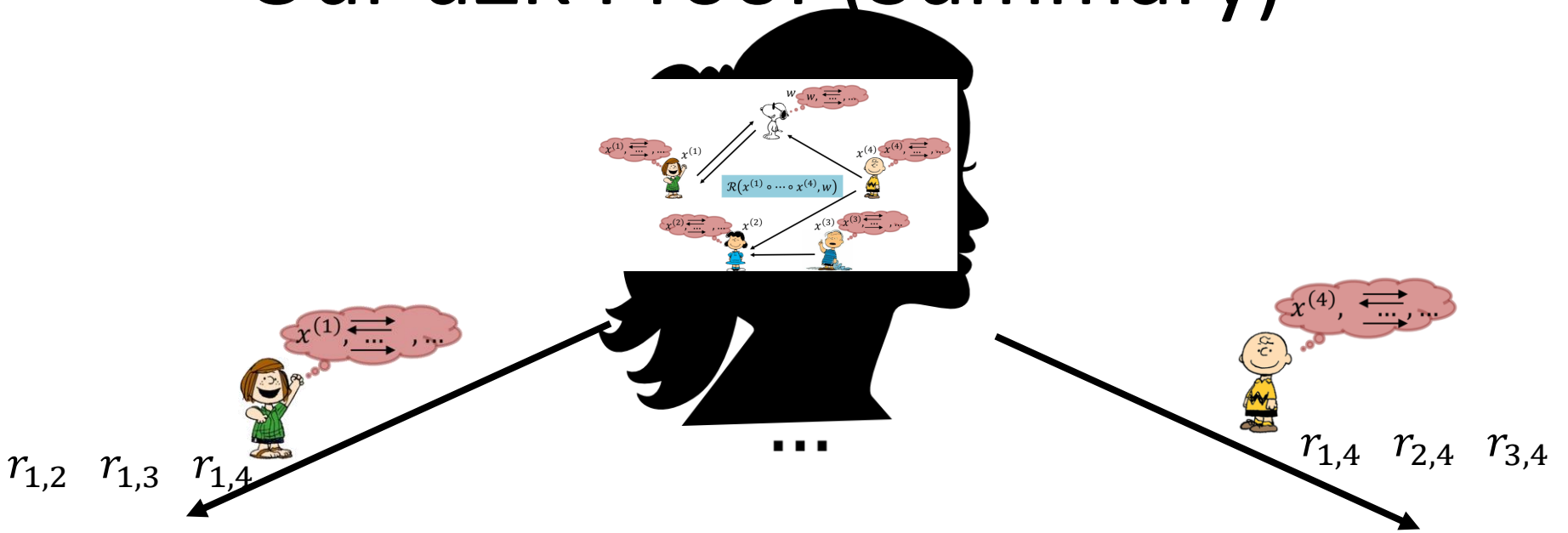
$t_{1,2} + r_{1,2}, t_{1,3} + r_{1,3}, t_{1,4} + r_{1,4}$
$view_4$ consistent?

- C_1 := set of verifiers claiming local inconsistency
- C_2 := set of verifiers b\casting incorrect messages (P computes and b\casts)
- V_i accepts if $|C_1 \cup C_2| \leq t$, otherwise rejects

Our dZK Proof (Summary)

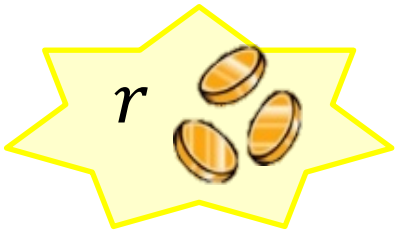
$V_1^{x^{(1)}}$ $V_2^{x^{(2)}}$ $V_3^{x^{(3)}}$ $V_4^{x^{(4)}}$

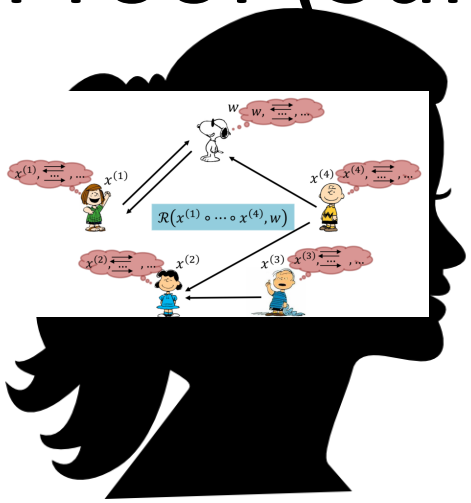
Our dZK Proof (Summary)



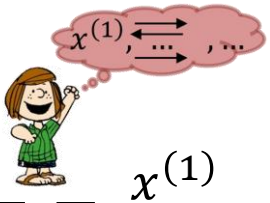
$V_1^{x^{(1)}}$ $V_2^{x^{(2)}}$ $V_3^{x^{(3)}}$ $V_4^{x^{(4)}}$

Our dZK Proof (Summary)

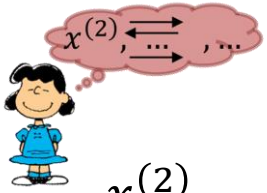




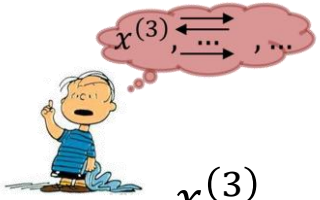
$r_{1,2}$ $r_{1,3}$ $r_{1,4}$



V_1 $x^{(1)}$

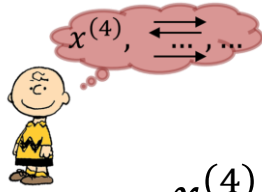


V_2 $x^{(2)}$



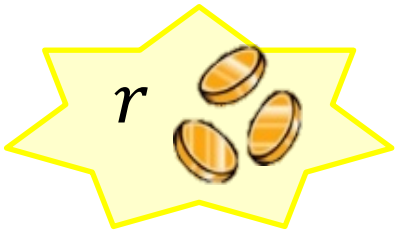
V_3 $x^{(3)}$

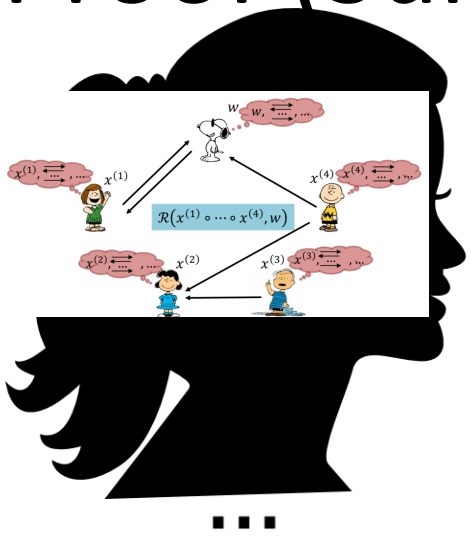
$r_{1,4}$ $r_{2,4}$ $r_{3,4}$



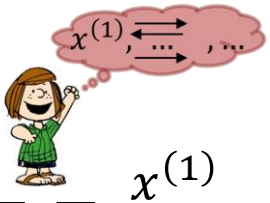
V_4 $x^{(4)}$

Our dZK Proof (Summary)





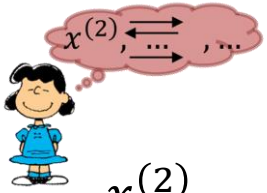
$r_{1,2}$ $r_{1,3}$ $r_{1,4}$



$x^{(1)}$

V_1

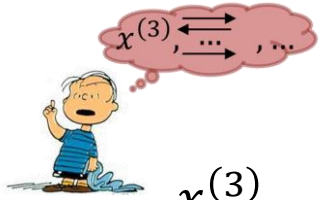
$t_{1,2} + r_{1,2}$
$t_{1,3} + r_{1,3}$
$t_{1,4} + r_{1,4}$
$view_1$ consistent?



$x^{(2)}$

V_2

$t_{2,1} + r_{1,2}$
$t_{2,3} + r_{2,3}$
$t_{2,4} + r_{2,4}$
$view_2$ consistent?

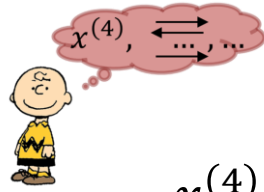


$x^{(3)}$

V_3

$t_{3,1} + r_{1,3}$
$t_{3,2} + r_{2,3}$
$t_{3,4} + r_{3,4}$
$view_3$ consistent?

$r_{1,4}$ $r_{2,4}$ $r_{3,4}$



$x^{(4)}$

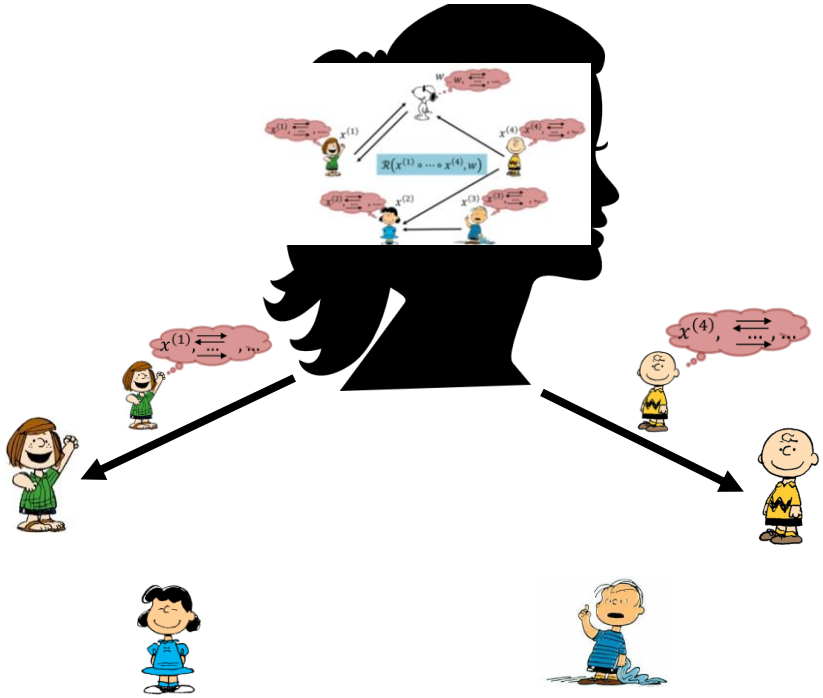
V_4

$t_{4,1} + r_{1,4}$
$t_{4,2} + r_{2,4}$
$t_{4,3} + r_{3,4}$
$view_4$ consistent?

MPC in the Head: Distributed vs. 2-Party

MPC in the Head: Distributed vs. 2-Party

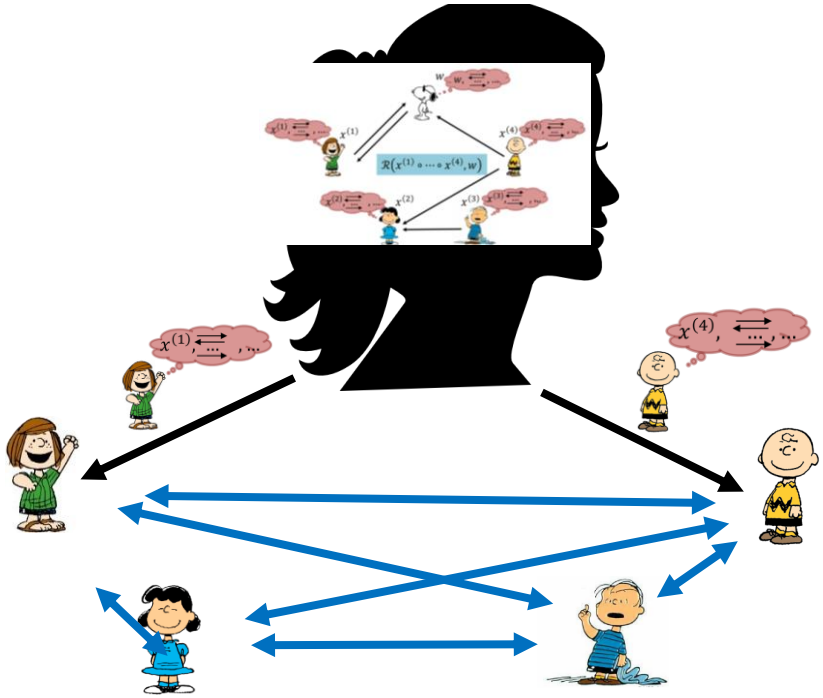
Distributed



MPC parties correspond to verifiers

MPC in the Head: Distributed vs. 2-Party

Distributed

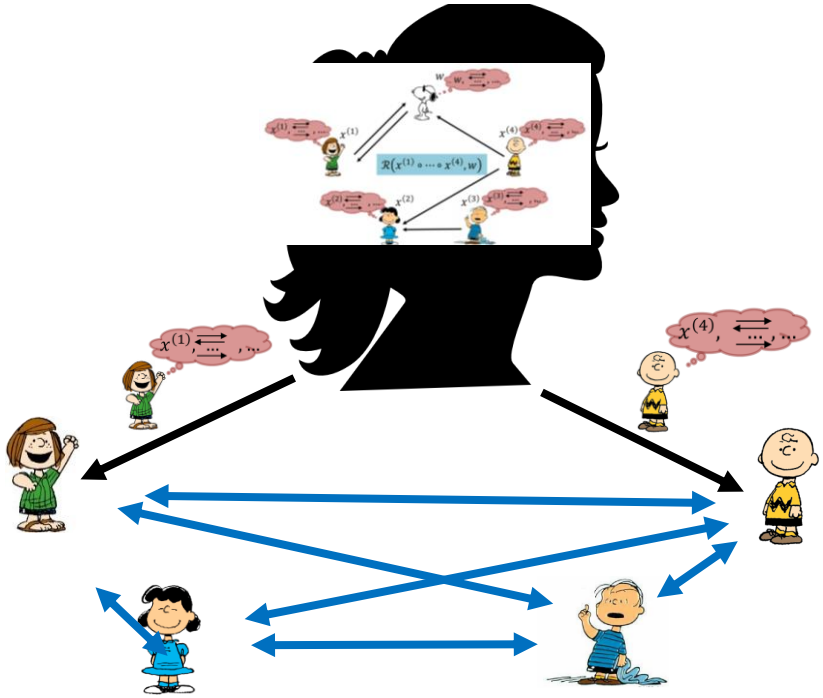


MPC parties correspond to verifiers

Entire execution trace checked

MPC in the Head: Distributed vs. 2-Party

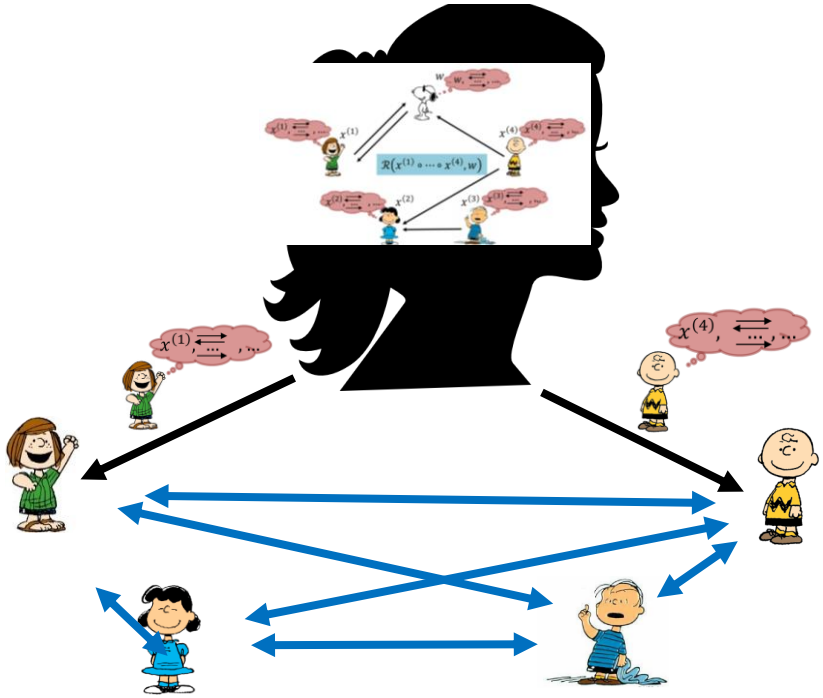
Distributed



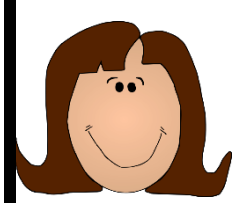
MPC parties correspond to verifiers
Entire execution trace checked
$negl(s)$ error, independent of # MPC parties

MPC in the Head: Distributed vs. 2-Party

Distributed



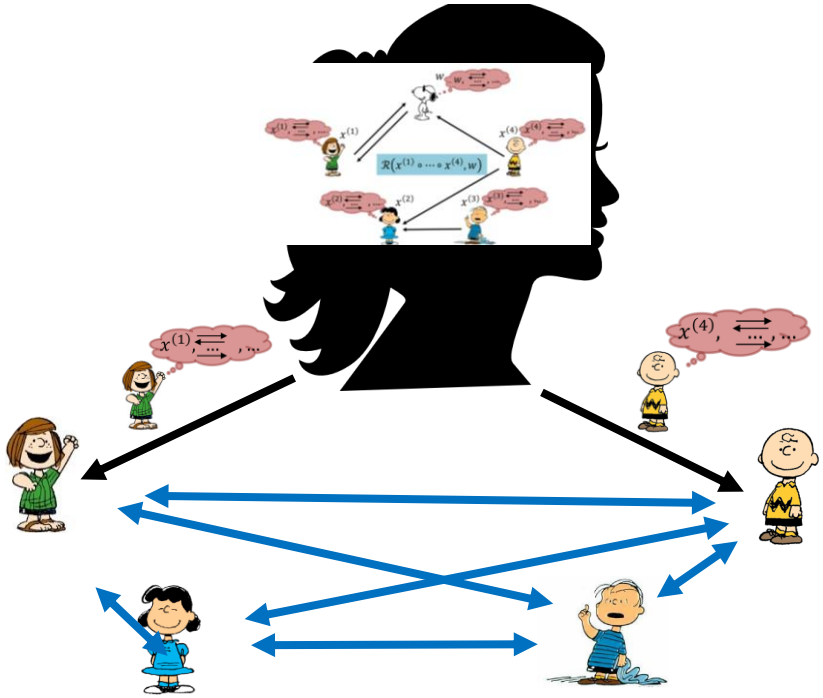
2-Party [IKOS07]



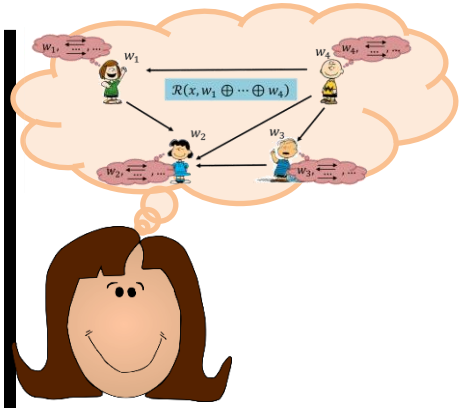
MPC parties correspond to verifiers
Entire execution trace checked
$negl(s)$ error, independent of # MPC parties

MPC in the Head: Distributed vs. 2-Party

Distributed



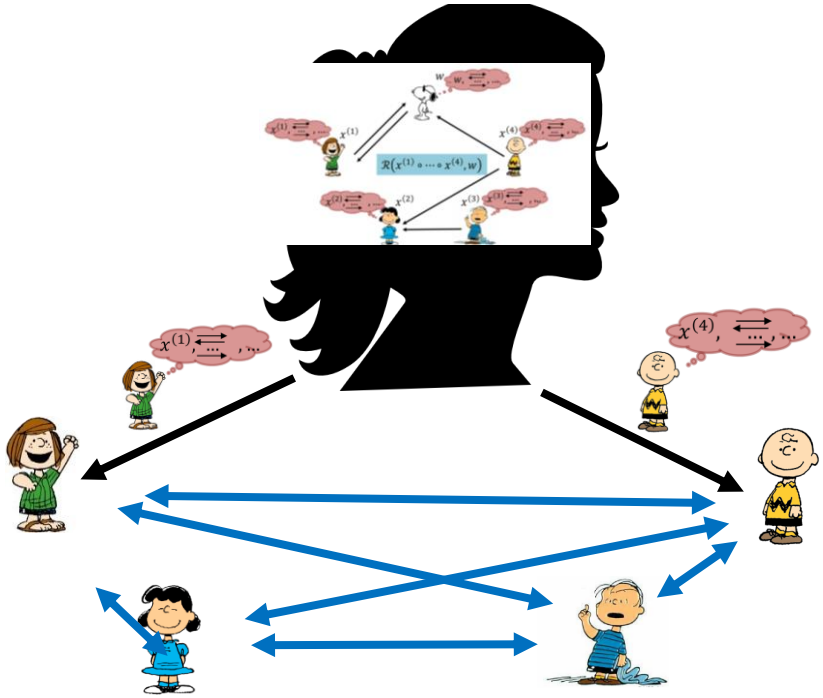
2-Party [IKOS07]



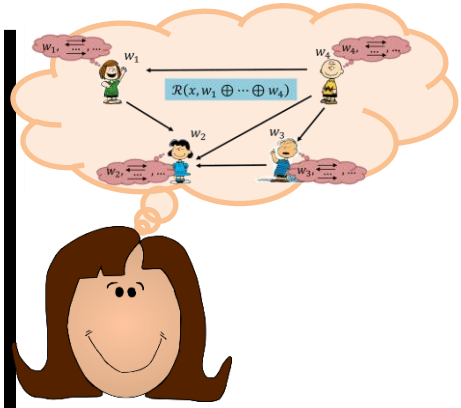
MPC parties correspond to verifiers	MPC parties are virtual
Entire execution trace checked	
$negl(s)$ error, independent of # MPC parties	

MPC in the Head: Distributed vs. 2-Party

Distributed



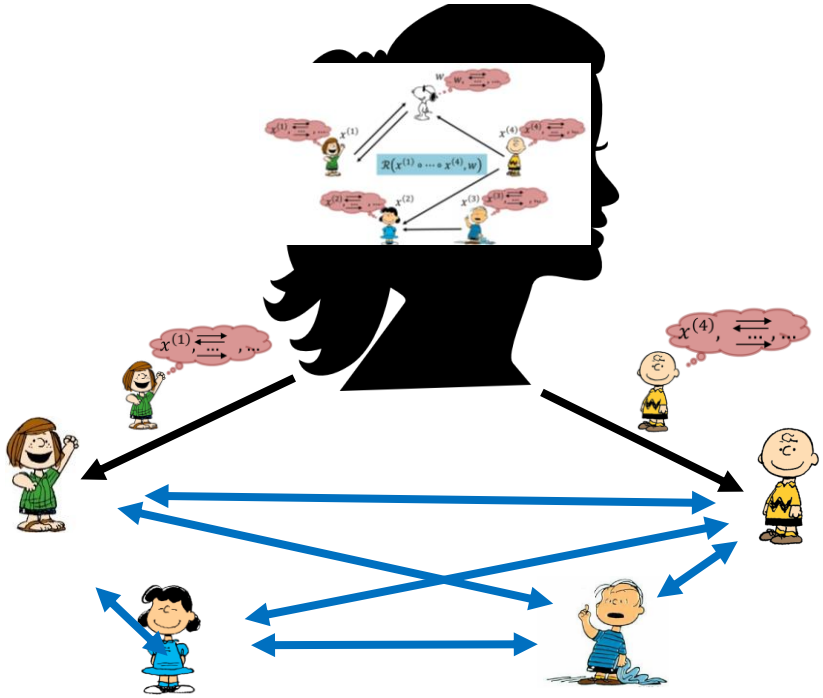
2-Party [IKOS07]



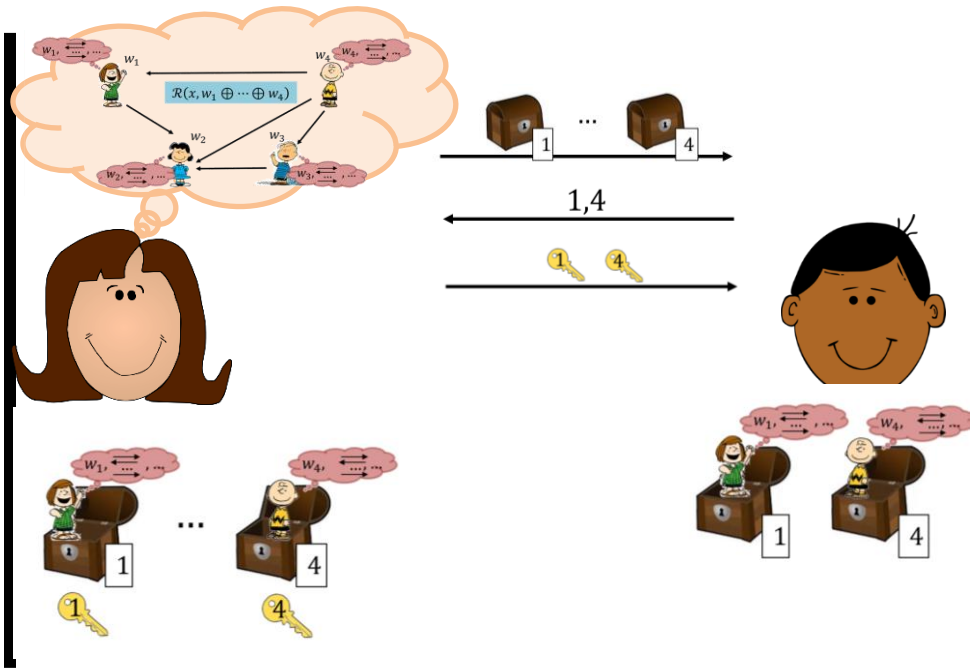
MPC parties correspond to verifiers	MPC parties are virtual
Entire execution trace checked	
$negl(s)$ error, independent of # MPC parties	$negl(s)$ error with s MPC parties

MPC in the Head: Distributed vs. 2-Party

Distributed



2-Party [IKOS07]



MPC parties correspond to verifiers	MPC parties are virtual
Entire execution trace checked	Partial execution trace checked
$negl(s)$ error, independent of # MPC parties	$negl(s)$ error with s MPC parties

Summary

- We saw **verification-efficient** **framing-free** dZK proofs
  - Based on a distributed version of “MPC in the head”

Summary

- We saw **verification-efficient** **framing-free** dZK proofs
  - Based on a distributed version of “MPC in the head”

- **More in the paper… (ePrint 2022/1523)**

Summary

- We saw **verification-efficient** **framing-free** dZK proofs
  - Based on a distributed version of "MPC in the head"

- **More in the paper… (ePrint 2022/1523)**
- Instantiations (assuming ideal coin toss, $\Omega(k)$ corruptions):

# Rounds	Total proof length	Verification CC
3	$O(\log k \cdot \log\|C\| \cdot \|C\|)$	$O(k^2)$
4	$O(\|C\|)$	$O(k^2 + s)$, s statistical sec param

Summary

- We saw **verification-efficient** **framing-free** dZK proofs
  - Based on a distributed version of "MPC in the head"

- **More in the paper… (ePrint 2022/1523)**
- Instantiations (assuming ideal coin toss, $\Omega(k)$ corruptions):

# Rounds	Total proof length	Verification CC
3	$O(\log k \cdot \log\|C\| \cdot \|C\|)$	$O(k^2)$
4	$O(\|C\|)$	$O(k^2 + s)$, s statistical sec param

- **Applications:** we give dZK-based:
  - VSS, Certifiable VSS + reusable
  - Framing-free proofs on distributed data (secure aggregation)
  - Semi-honest to malicious compiler: with identifiable abort
- Framing-free property crucial for these applications

Summary

- We saw **verification-efficient** **framing-free** dZK proofs
  - Based on a distributed version of "MPC in the head"

- **More in the paper… (ePrint 2022/1523)**
- Instantiations (assuming ideal coin toss, $\Omega(k)$ corruptions):

# Rounds	Total proof length	Verification CC
3	$O(\log k \cdot \log\|C\| \cdot \|C\|)$	$O(k^2)$
4	$O(\|C\|)$	$O(k^2 + s)$, s statistical sec param

- **Applications:** we give dZK-based:
  - VSS, Certifiable VSS + reusable
  - Framing-free proofs on distributed data (secure aggregation)
  - Semi-honest to malicious compiler: with identifiable abort
- Framing-free property crucial for these applications

Thank you!