Zero-Knowledge Proofs

Lecture 15

Interactive Proofs

Interactive Proofs

Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property

Interactive Proofs

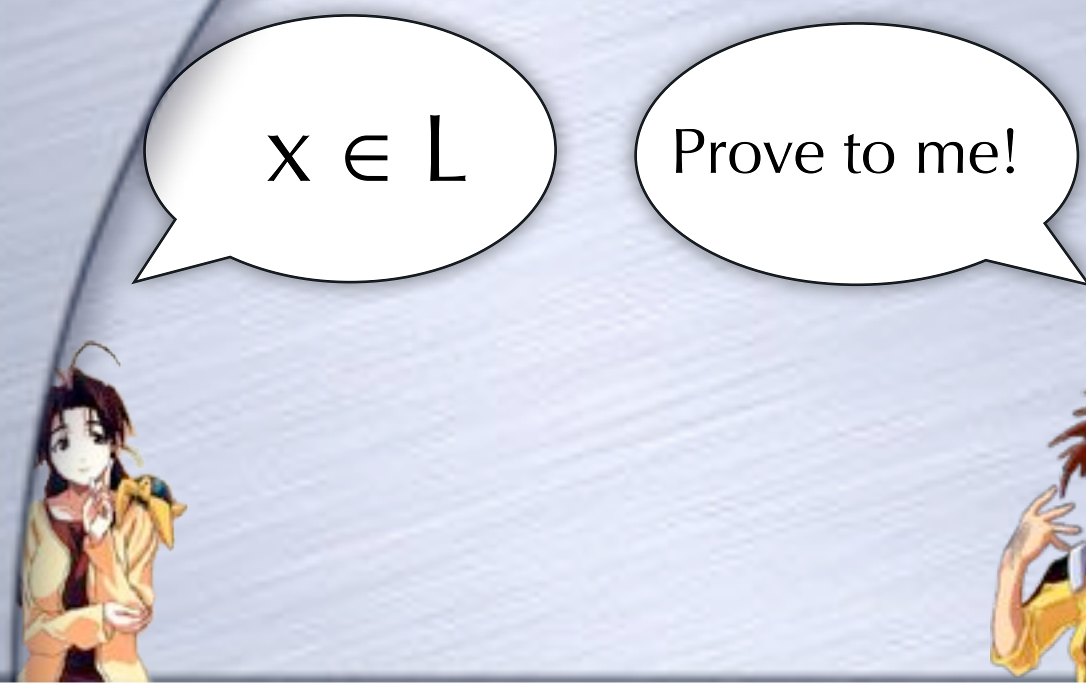
- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L

Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L

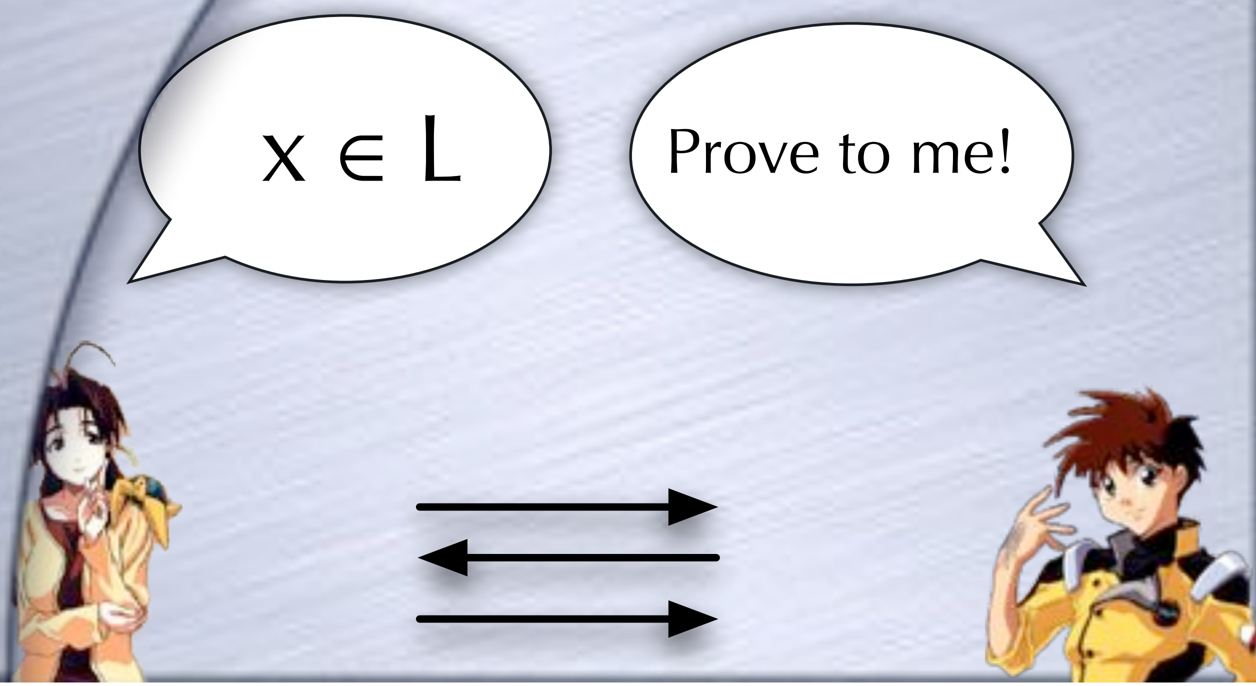
Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L



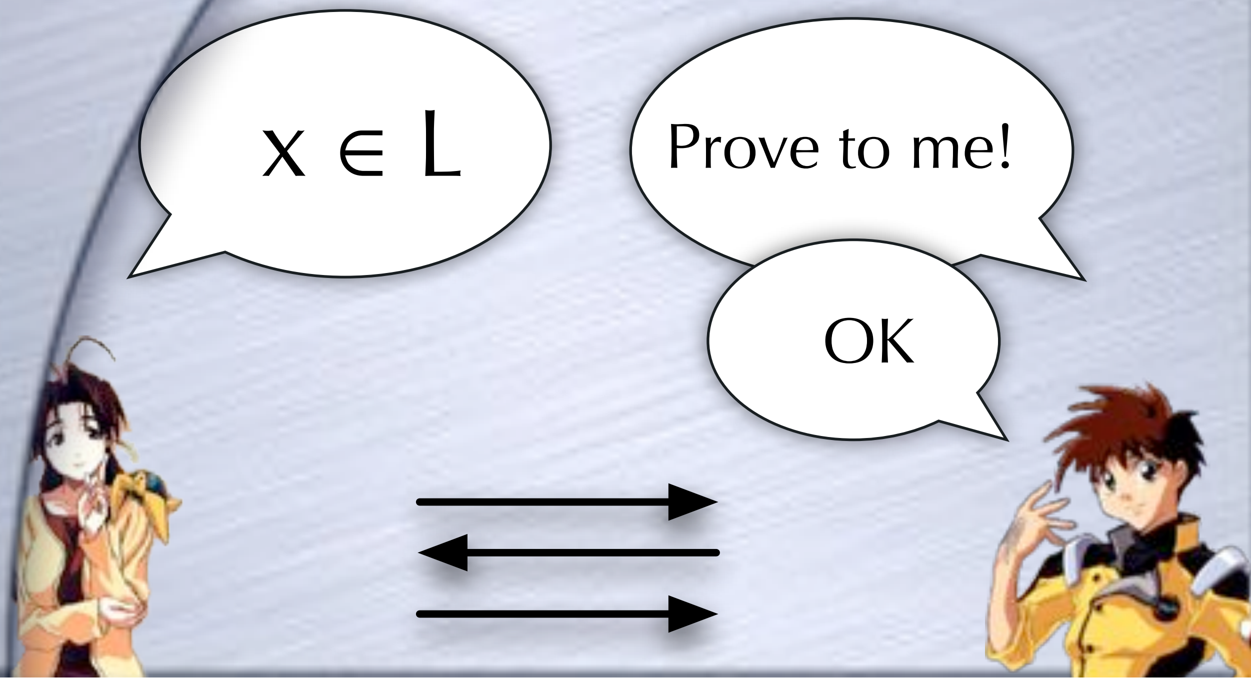
Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L



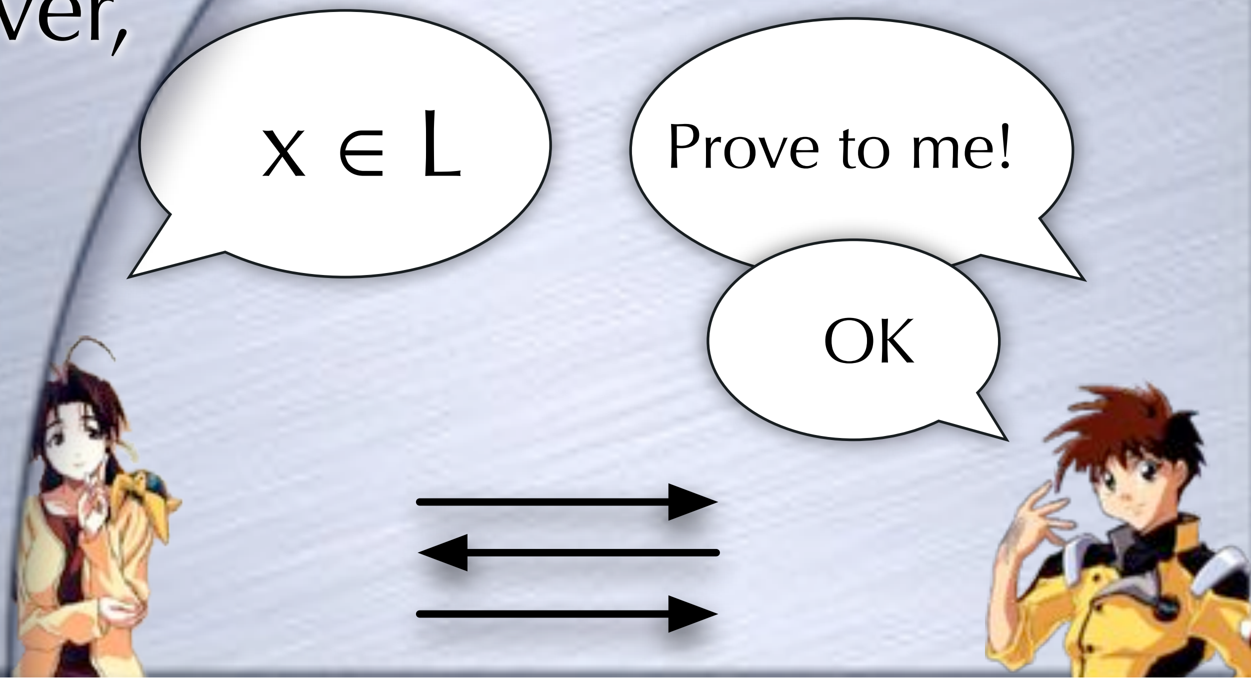
Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L



Interactive Proofs

- ***Prover*** wants to convince ***verifier*** that x has some property
  - i.e. x is in language L
- All powerful prover, computationally bounded verifier (for now)



Interactive Proofs

Interactive Proofs

- Completeness

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**


Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

Interactive Proofs

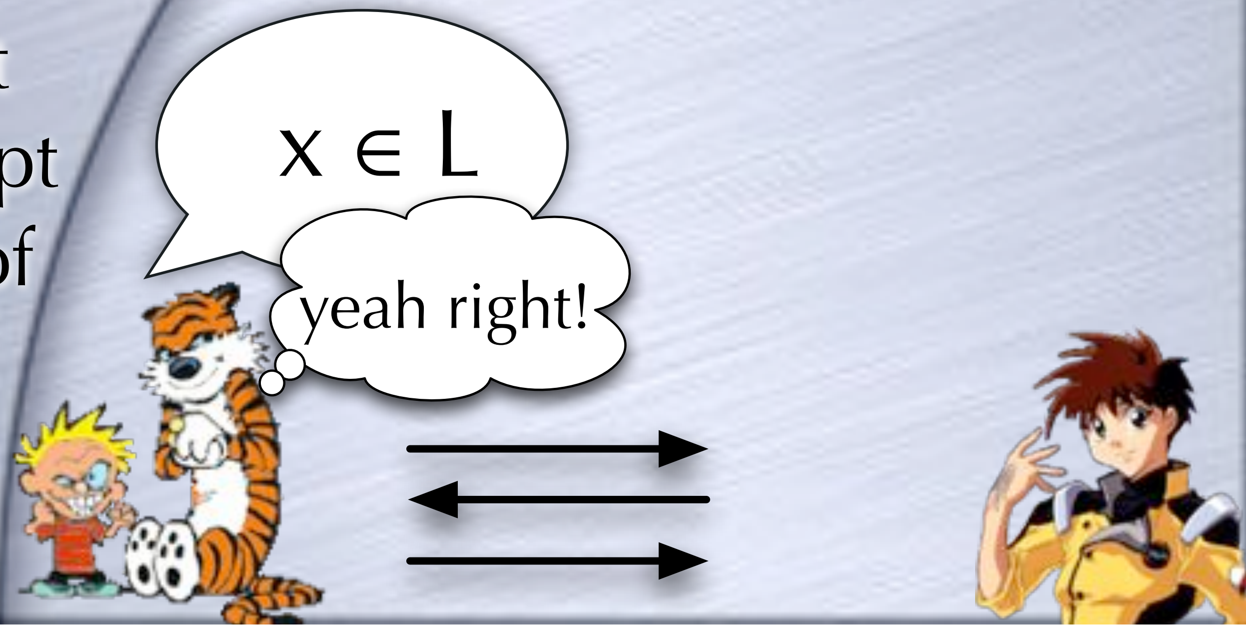
- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

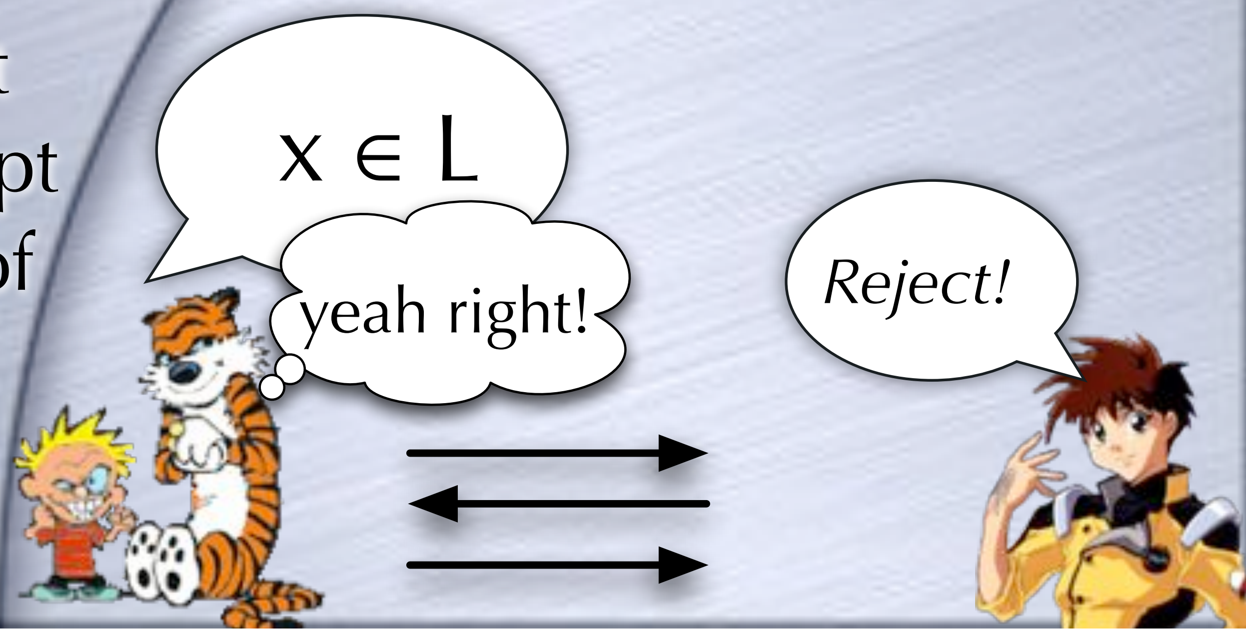
Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof



Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof



An Example

An Example

- Coke in bottle or can

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different

An Example

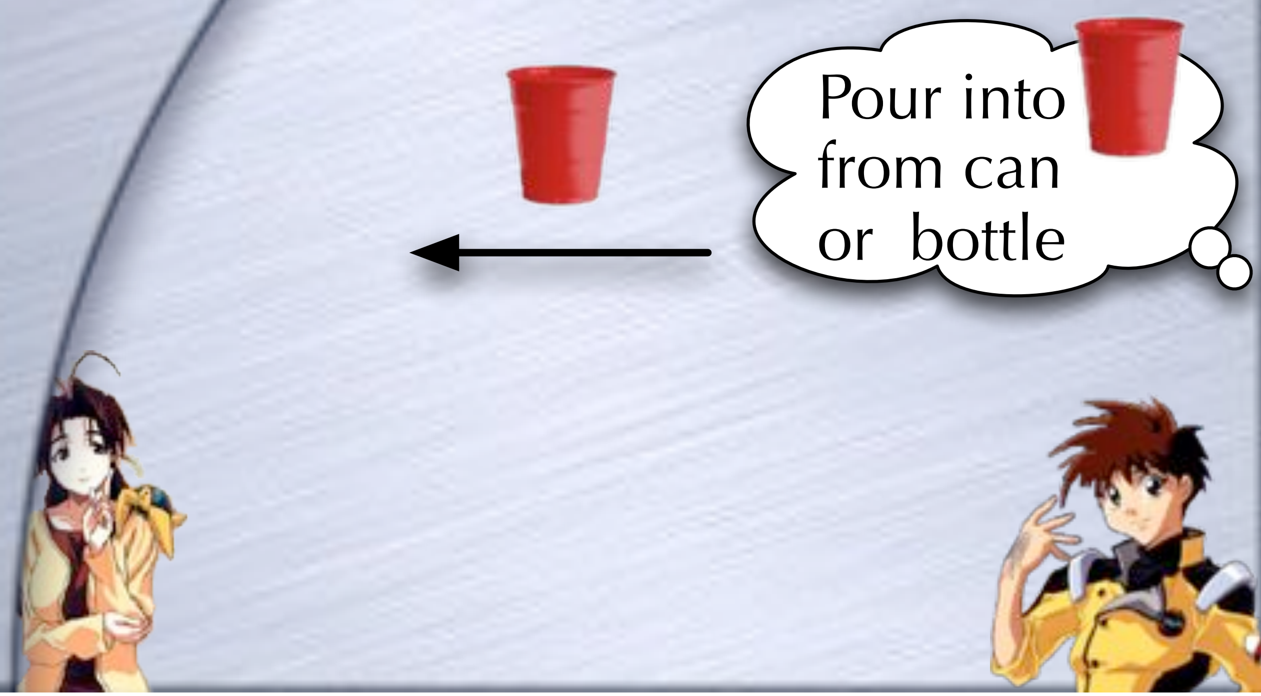
- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

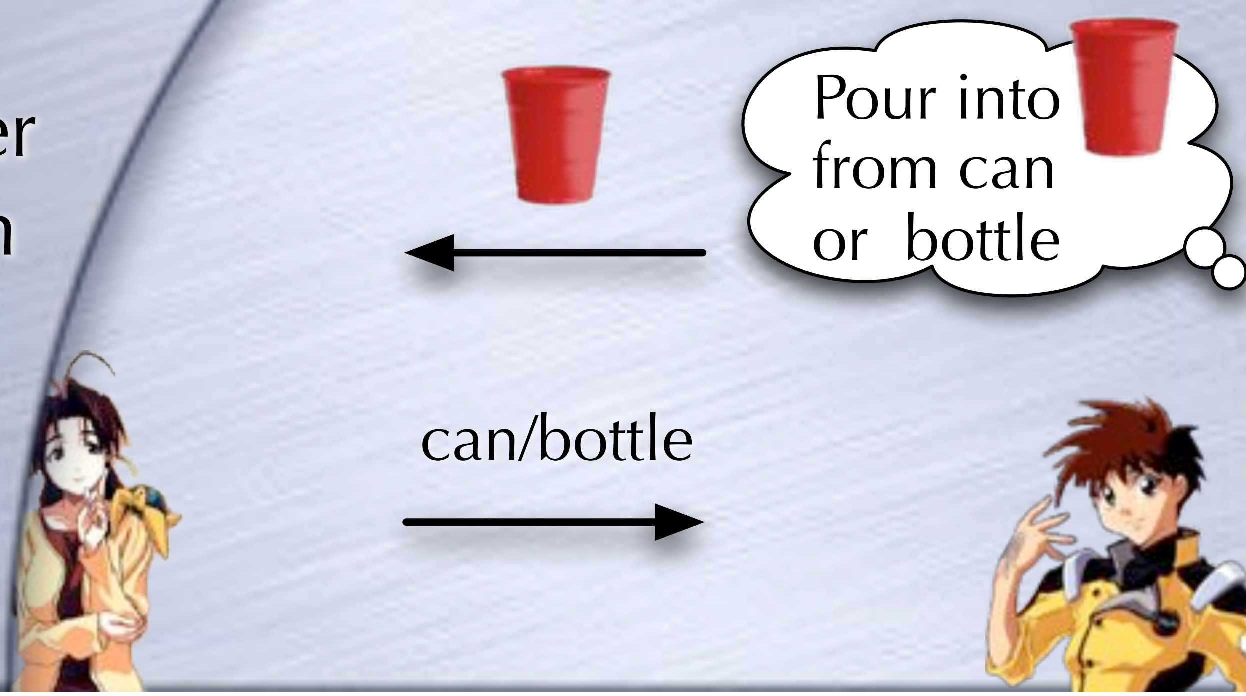


An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:
  - prover tells whether cup was filled from can or bottle

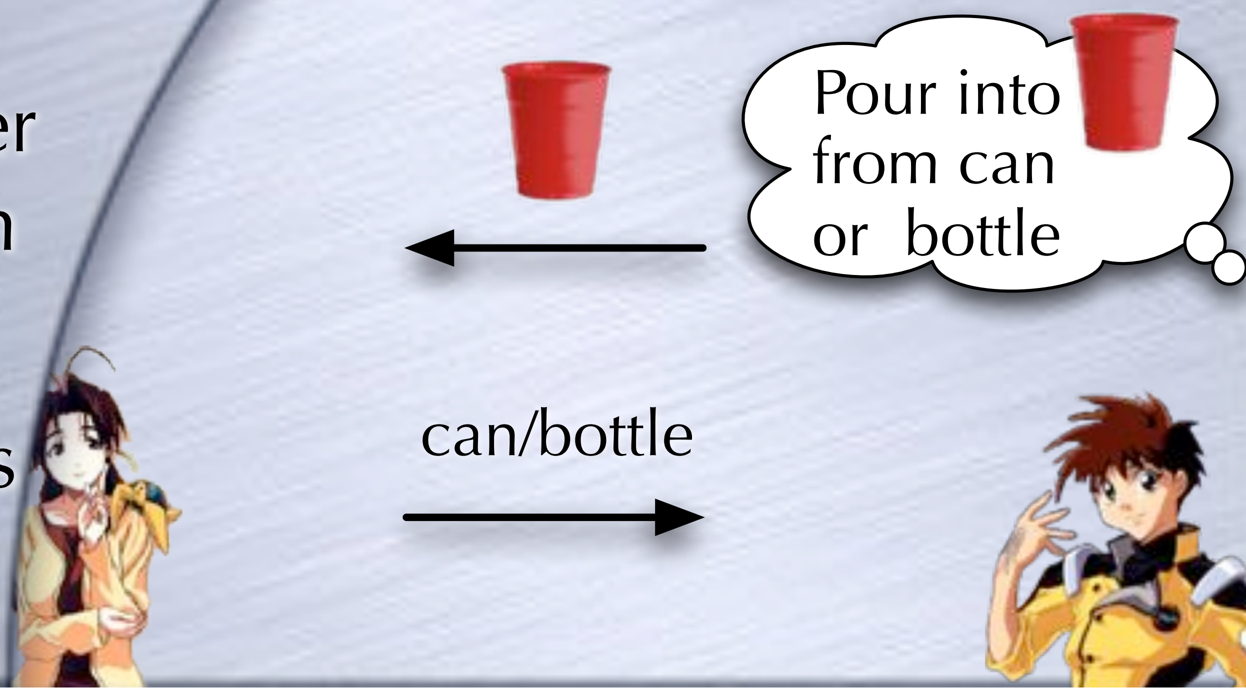
An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:
  - prover tells whether cup was filled from can or bottle



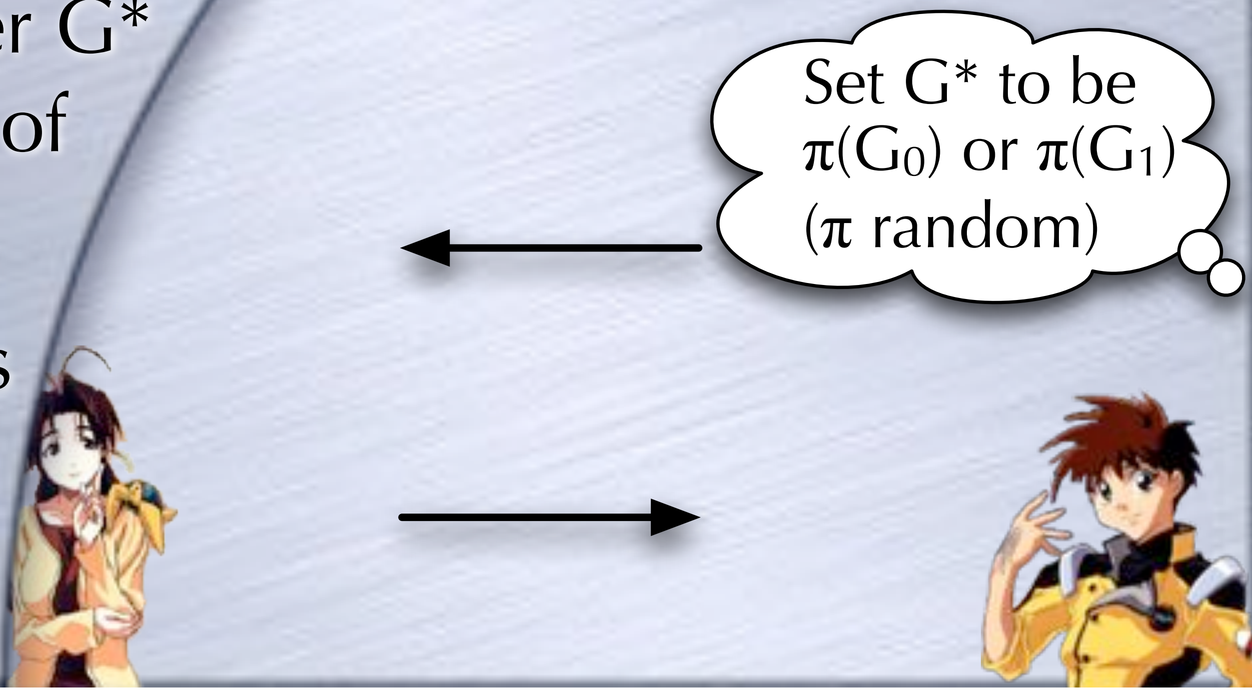
An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:
  - prover tells whether cup was filled from can or bottle
  - repeat till verifier is convinced



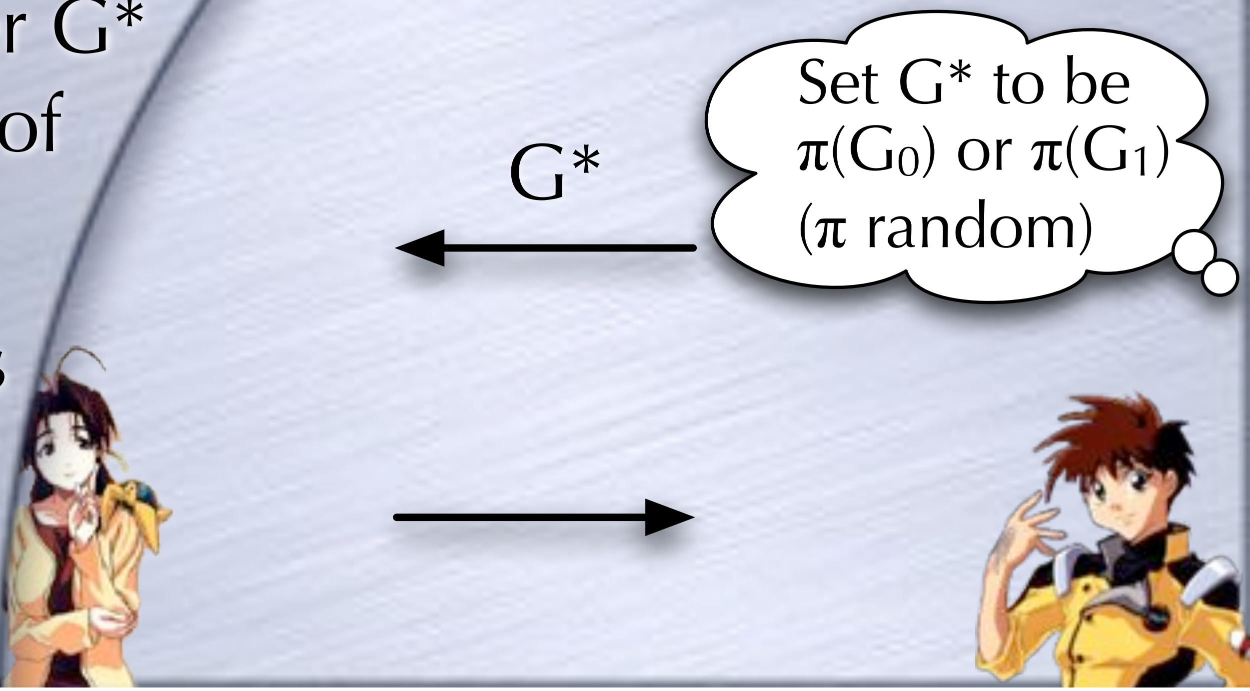
An Example

- **Graph Non-Isomorphism**
  - Prover claims: G_0 ***not*** isomorphic to G_1
- IP protocol:
  - prover tells whether G^* is an isomorphism of G_0 or G_1
  - repeat till verifier is convinced



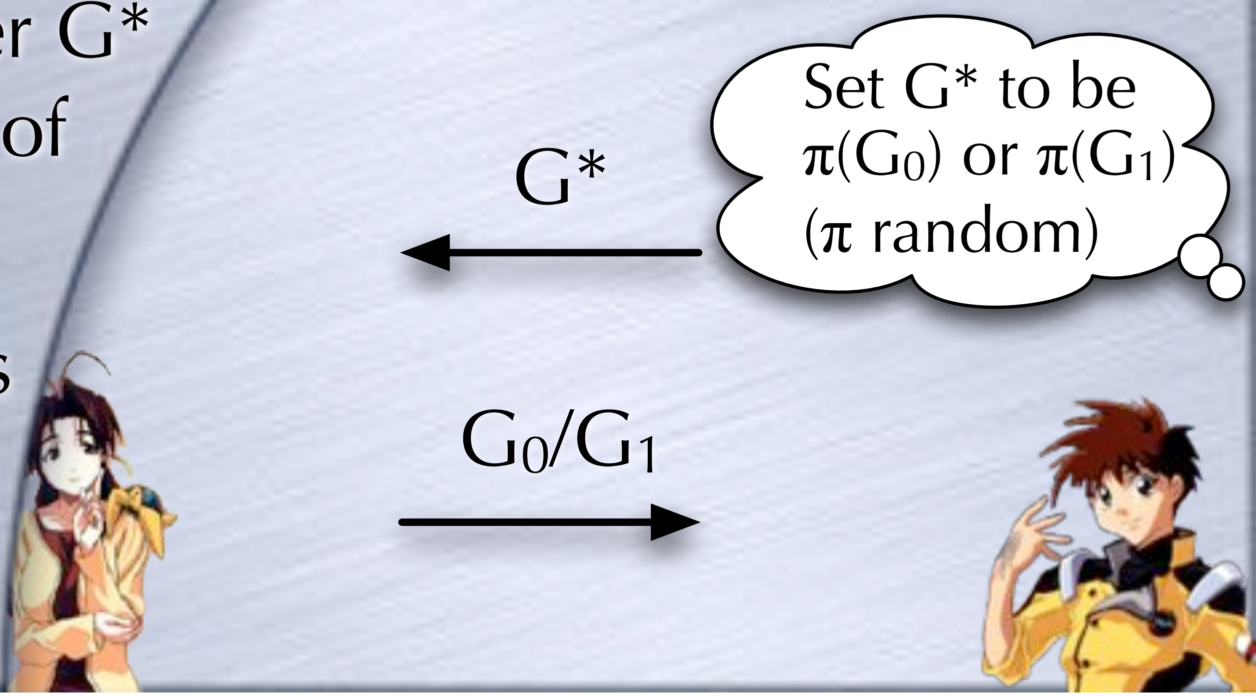
An Example

- **Graph Non-Isomorphism**
  - Prover claims: G_0 ***not*** isomorphic to G_1
- IP protocol:
  - prover tells whether G* is an isomorphism of G_0 or G_1
  - repeat till verifier is convinced

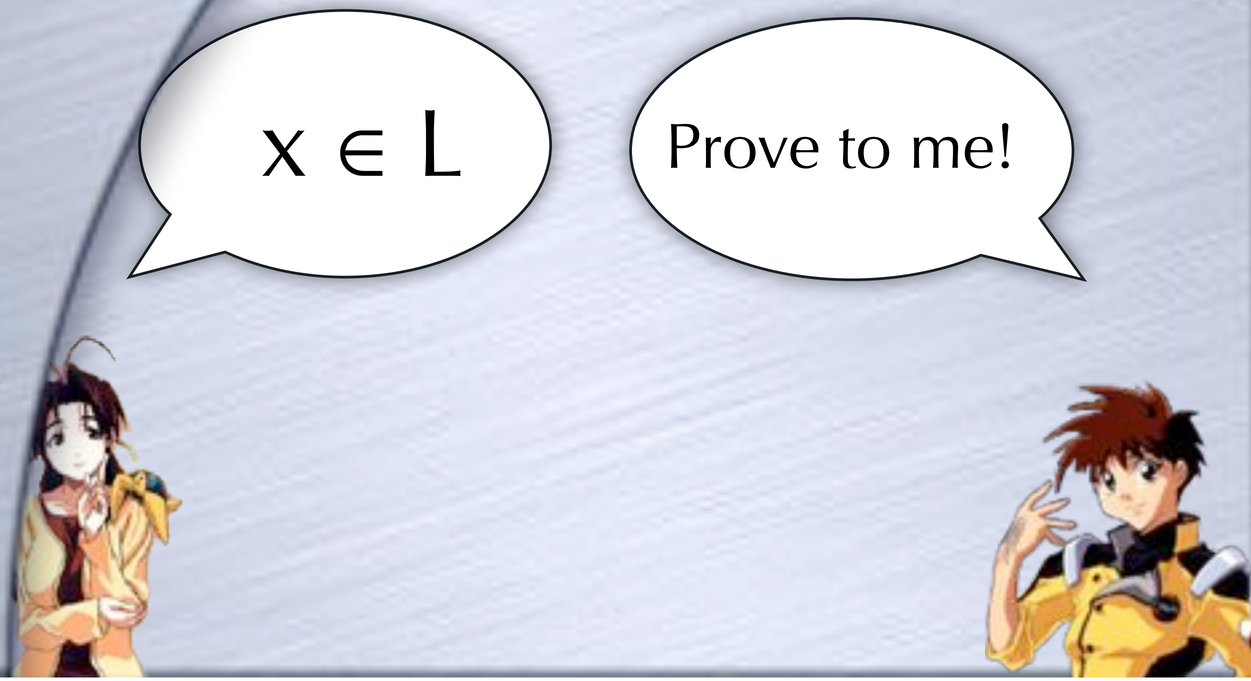


An Example

- **Graph Non-Isomorphism**
  - Prover claims: G_0 ***not*** isomorphic to G_1
- IP protocol:
  - prover tells whether G^* is an isomorphism of G_0 or G_1
  - repeat till verifier is convinced

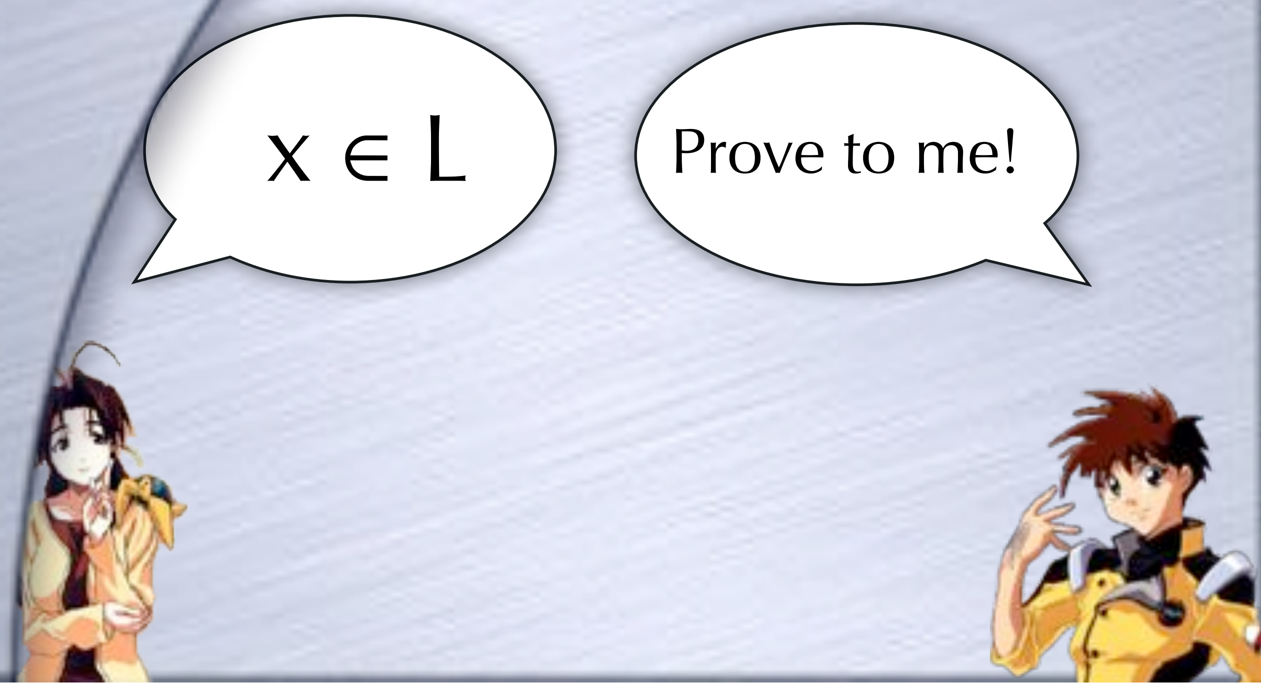


Proofs for NP languages



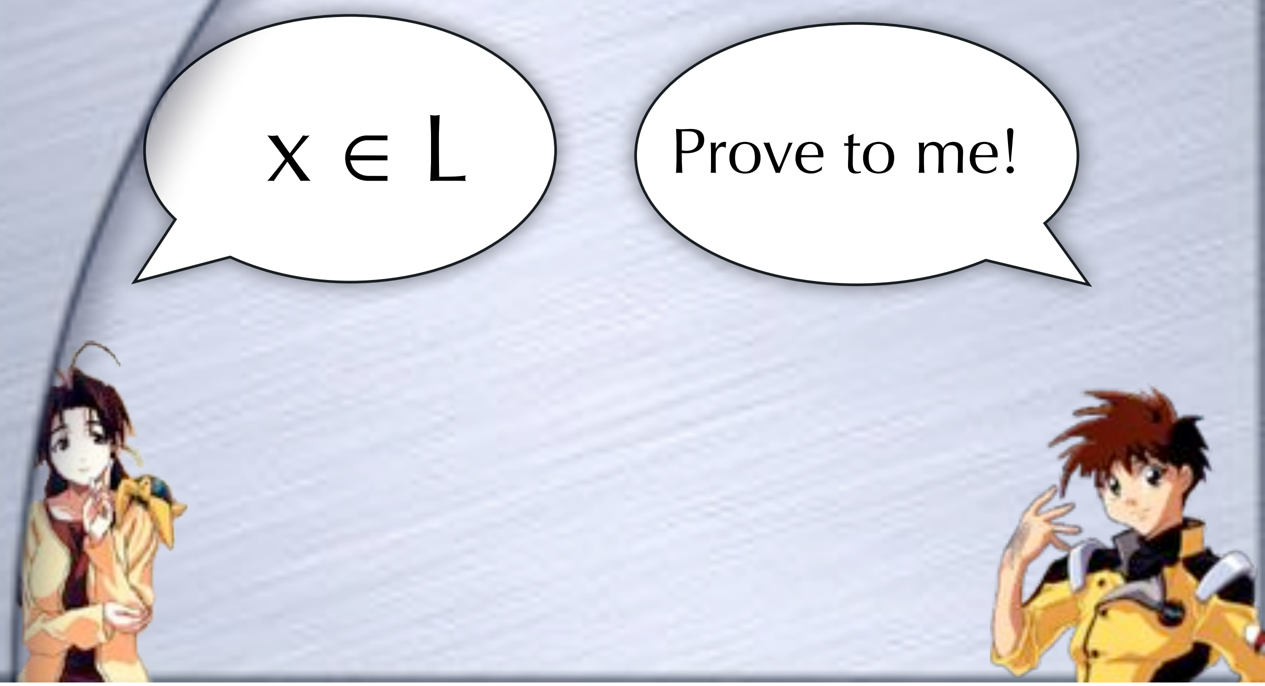
Proofs for NP languages

- Proving membership in an **NP** language L



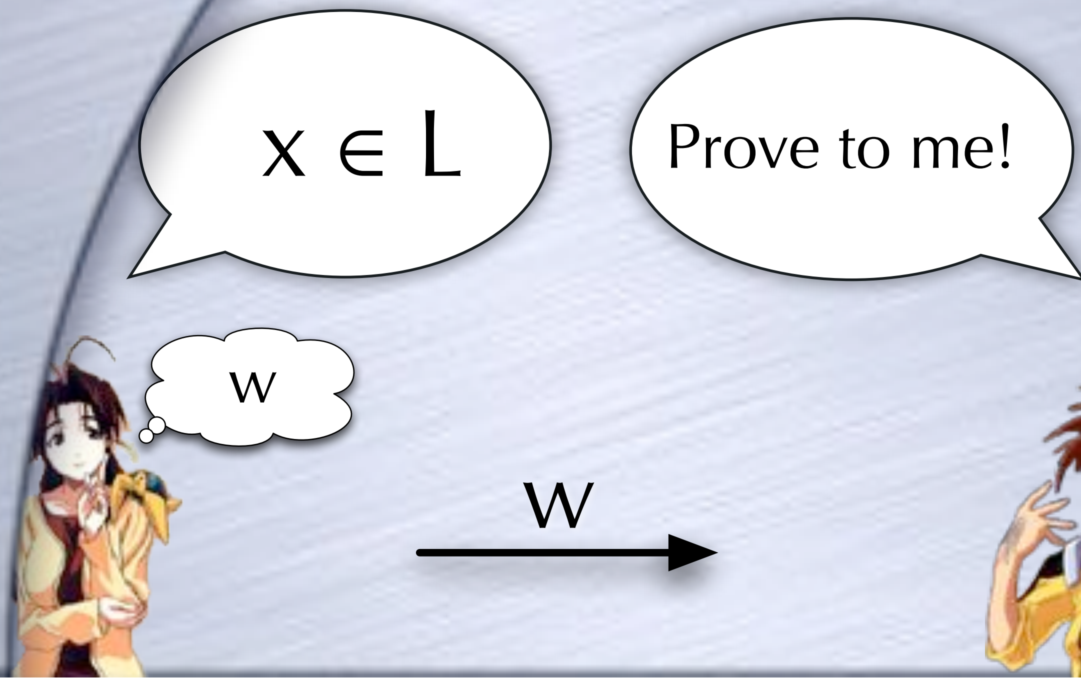
Proofs for NP languages

- Proving membership in an **NP** language L
  - $x \in L$ iff $\exists w\ R(x,w)=1$ (for R in **P**)



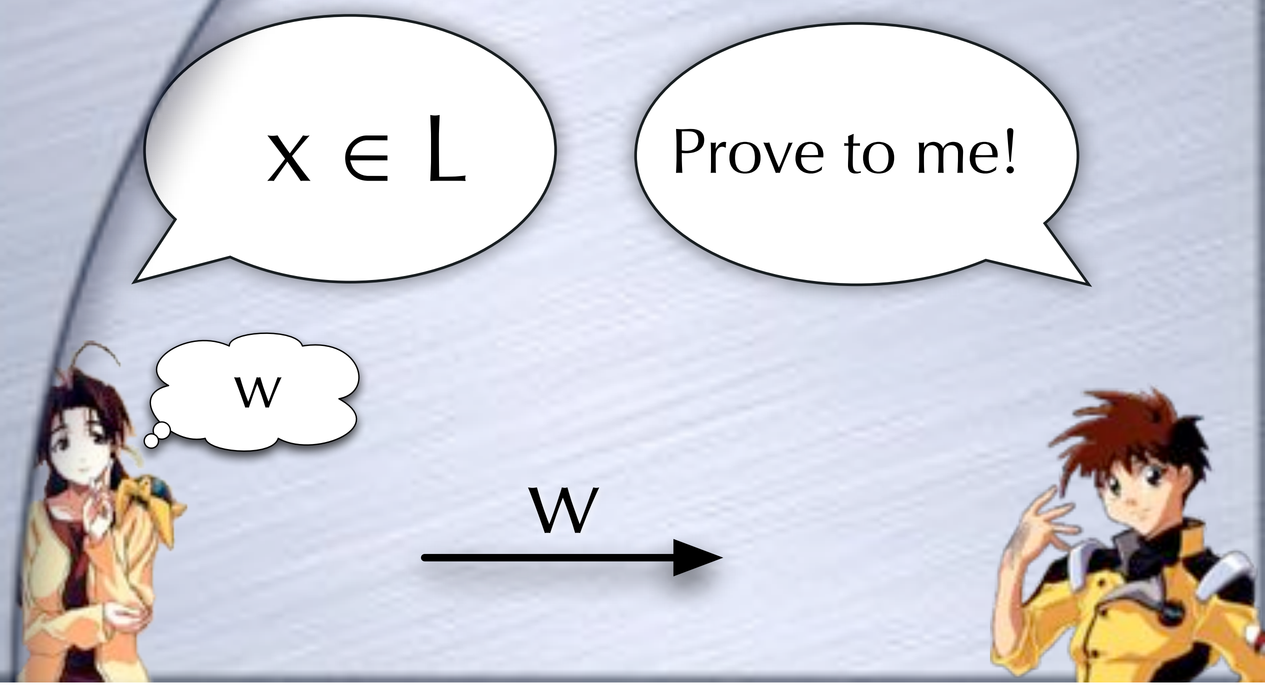
Proofs for NP languages

- Proving membership in an **NP** language L
  - x ∈ L iff ∃w R(x,w)=1(for R in **P**)
    - e.g. Graph Isomorphism



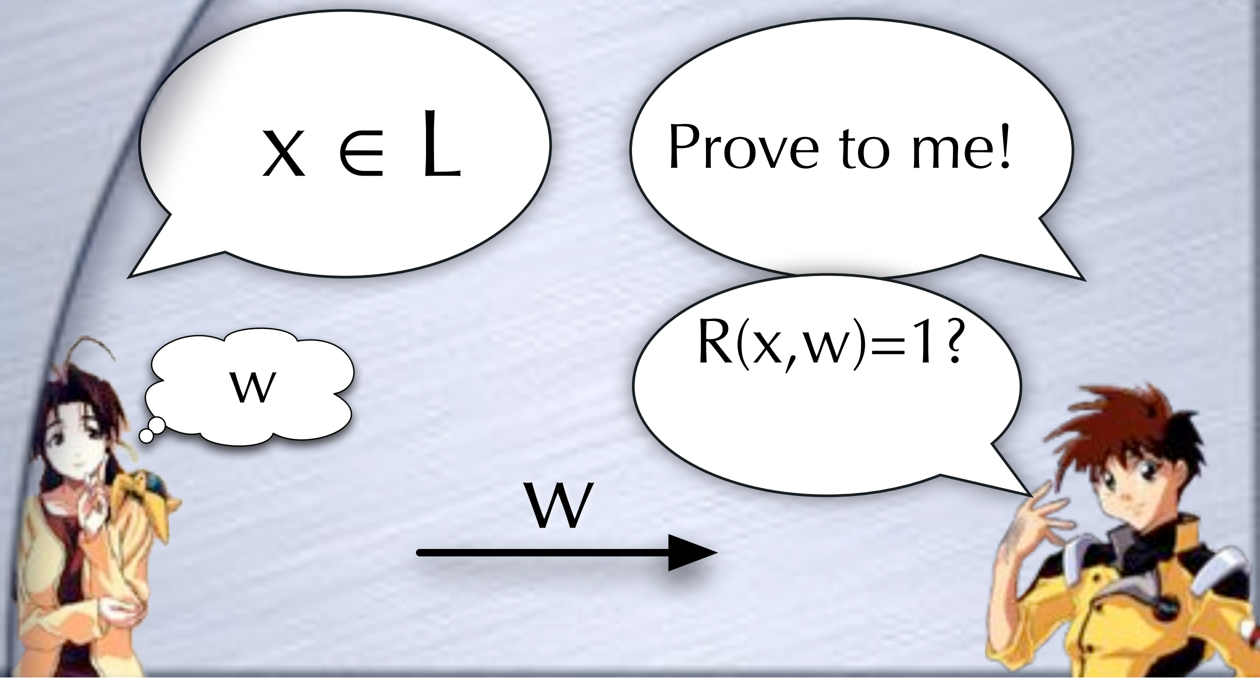
Proofs for NP languages

- Proving membership in an **NP** language L
  - $x \in L$ iff $\exists w$ $R(x,w)=1$ (for R in **P**)
    - e.g. Graph Isomorphism
- IP protocol:



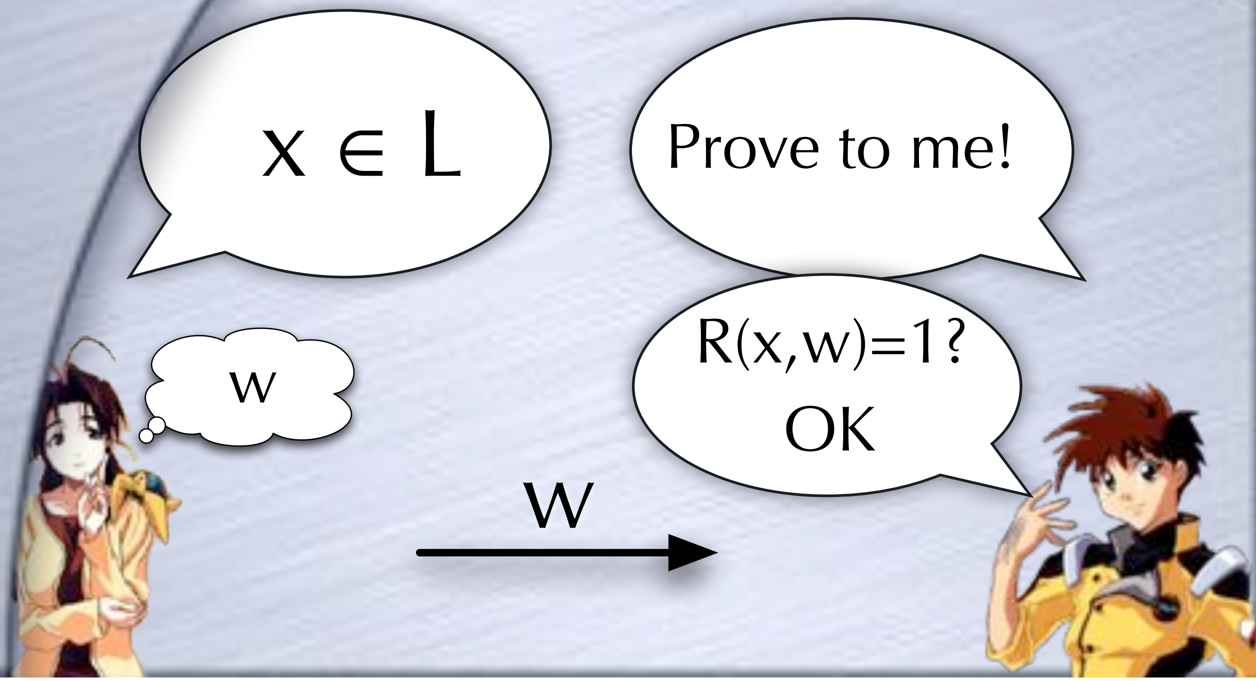
Proofs for NP languages

- Proving membership in an **NP** language L
  - x ∈ L iff ∃w R(x,w)=1 (for R in **P**)
    - e.g. Graph Isomorphism
- IP protocol:



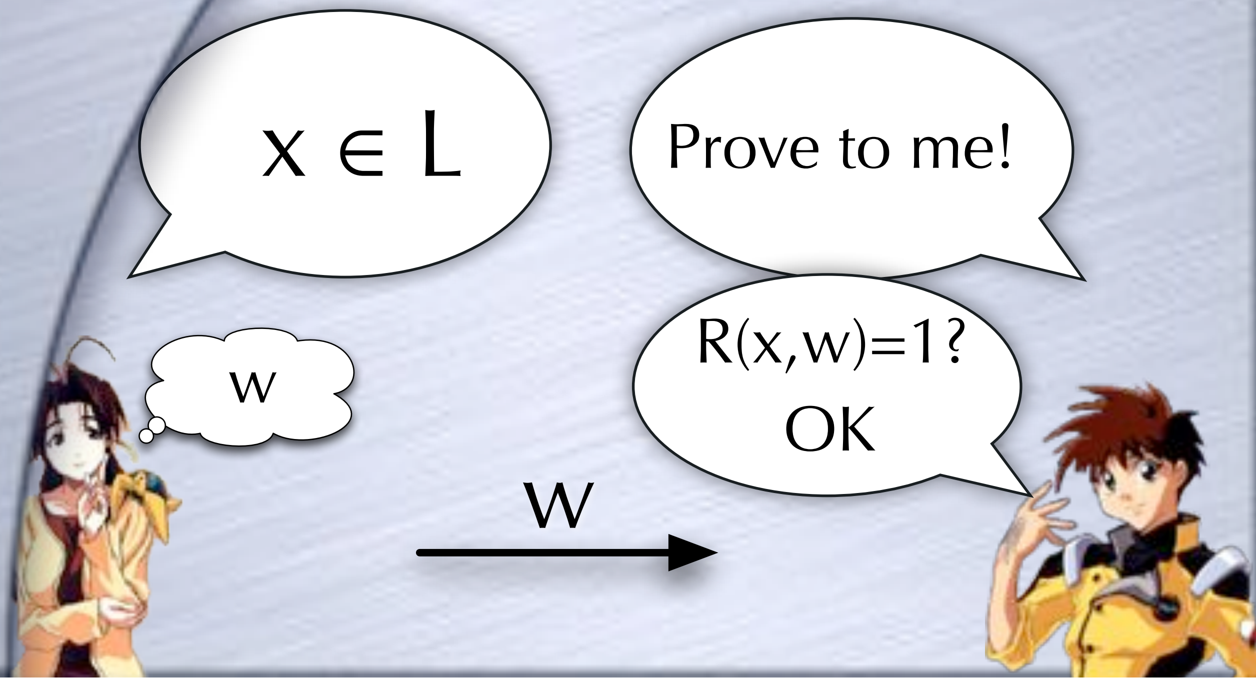
Proofs for NP languages

- Proving membership in an **NP** language L
  - x ∈ L iff ∃w R(x,w)=1(for R in **P**)
    - e.g. Graph Isomorphism
- IP protocol:



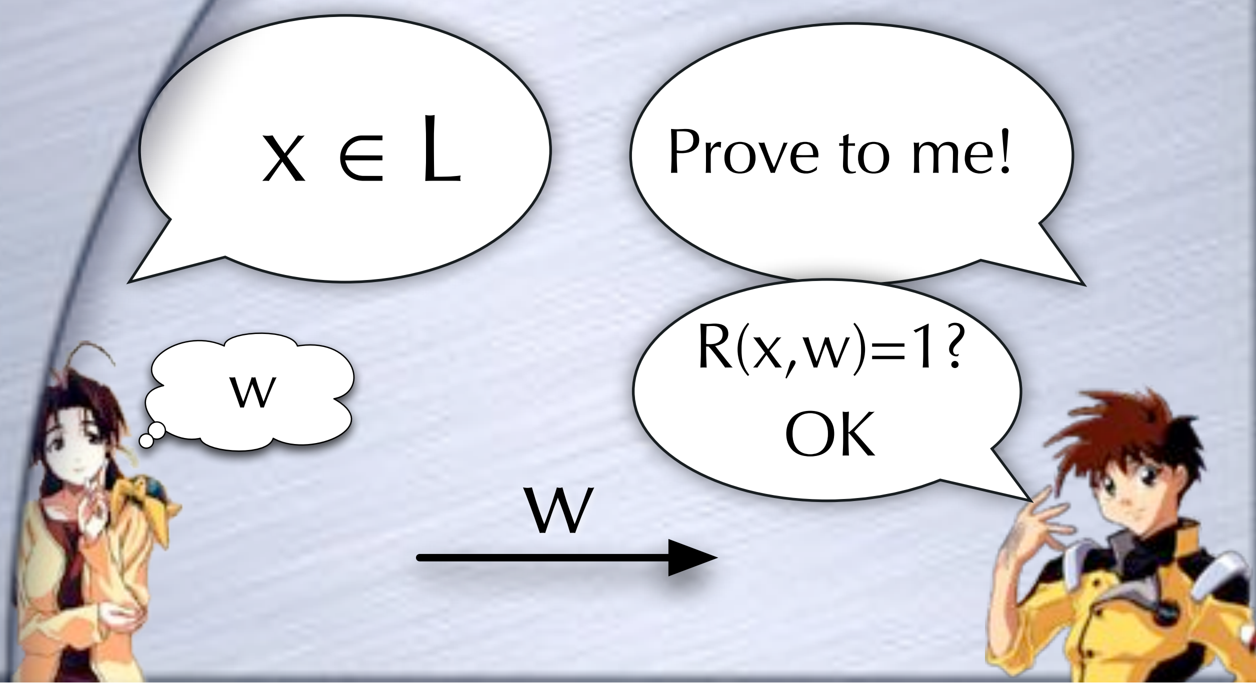
Proofs for NP languages

- Proving membership in an **NP** language L
  - $x \in L$ iff $\exists w\ R(x,w)=1$ (for R in **P**)
    - e.g. Graph Isomorphism
- IP protocol:
  - prover sends w (non-interactive)



Proofs for NP languages

- Proving membership in an **NP** language L
  - x ∈ L iff ∃w R(x,w)=1 (for R in **P**)
    - e.g. Graph Isomorphism
- IP protocol:
  - prover sends w (non-interactive)
- *What if prover doesn't want to reveal w?*



Zero-Knowledge Proofs

Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover

Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L

Zero-Knowledge Proofs

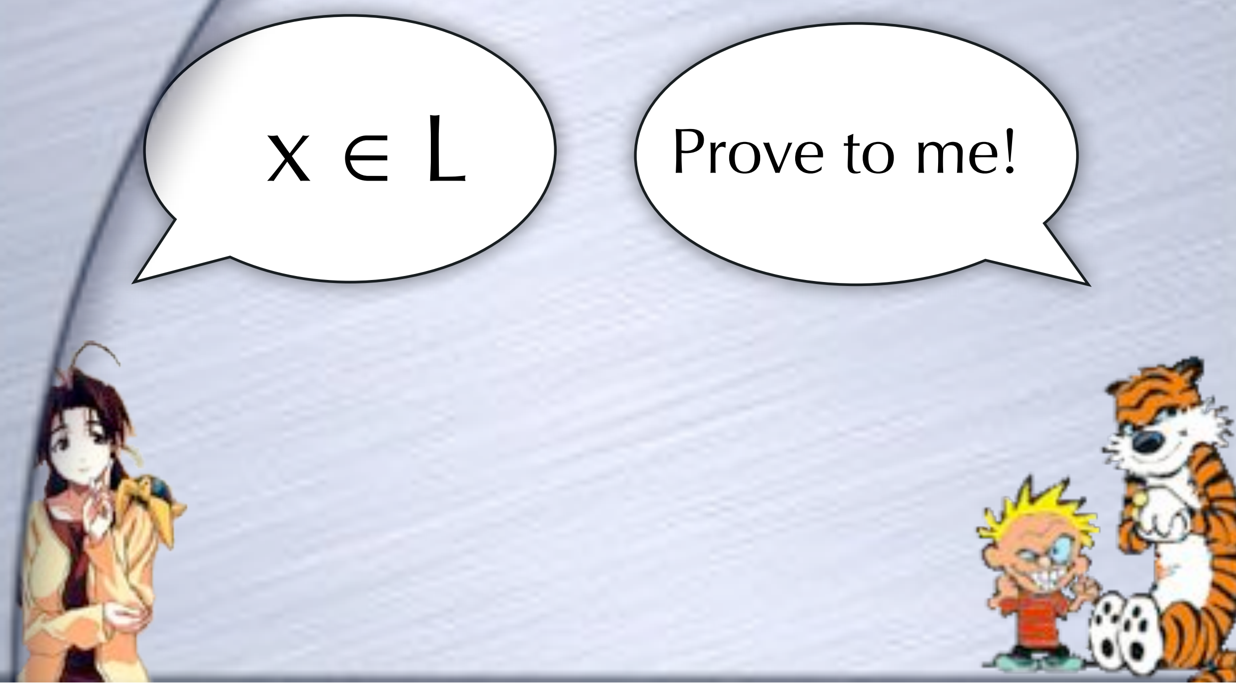
- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L

Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L

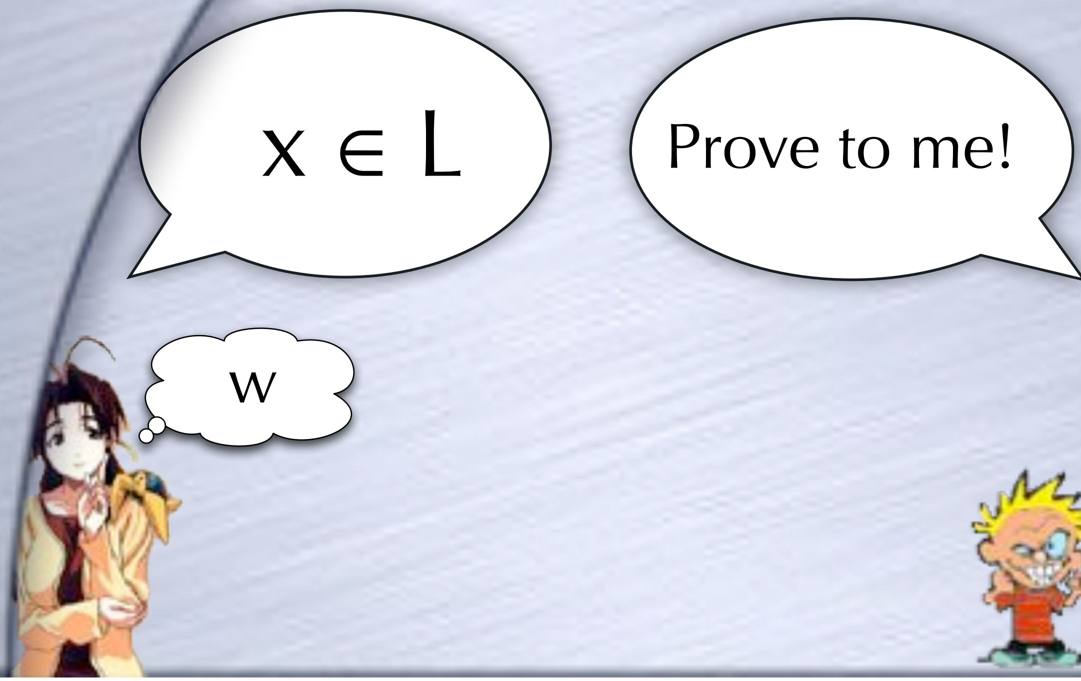
Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L



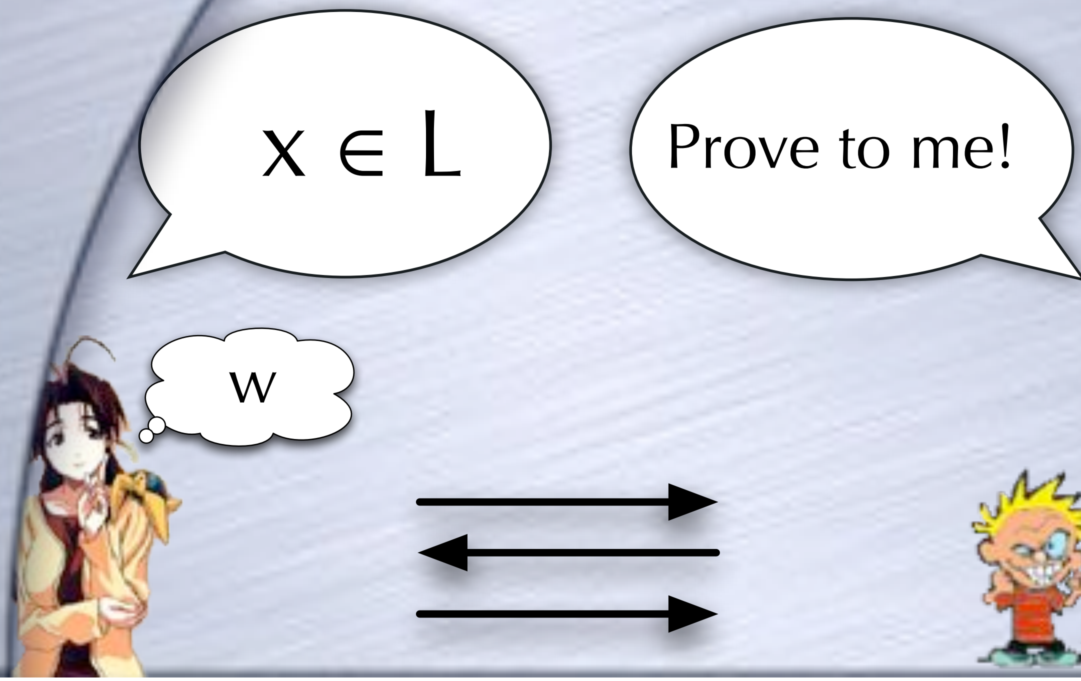
Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L



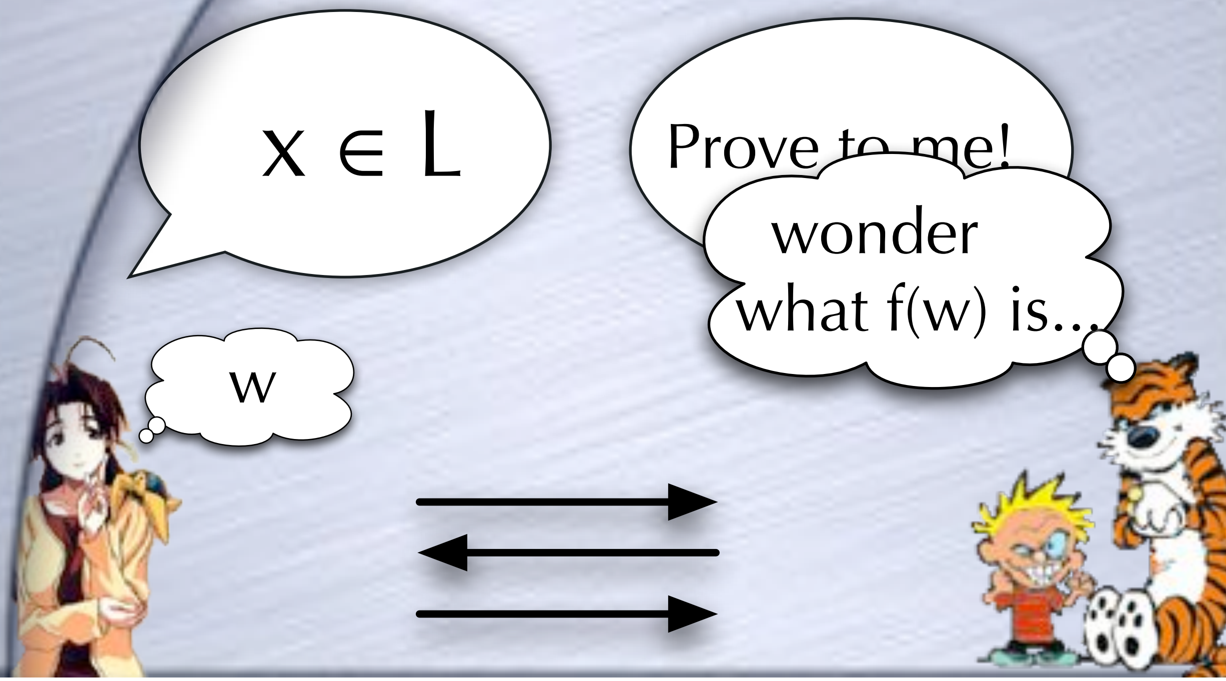
Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L



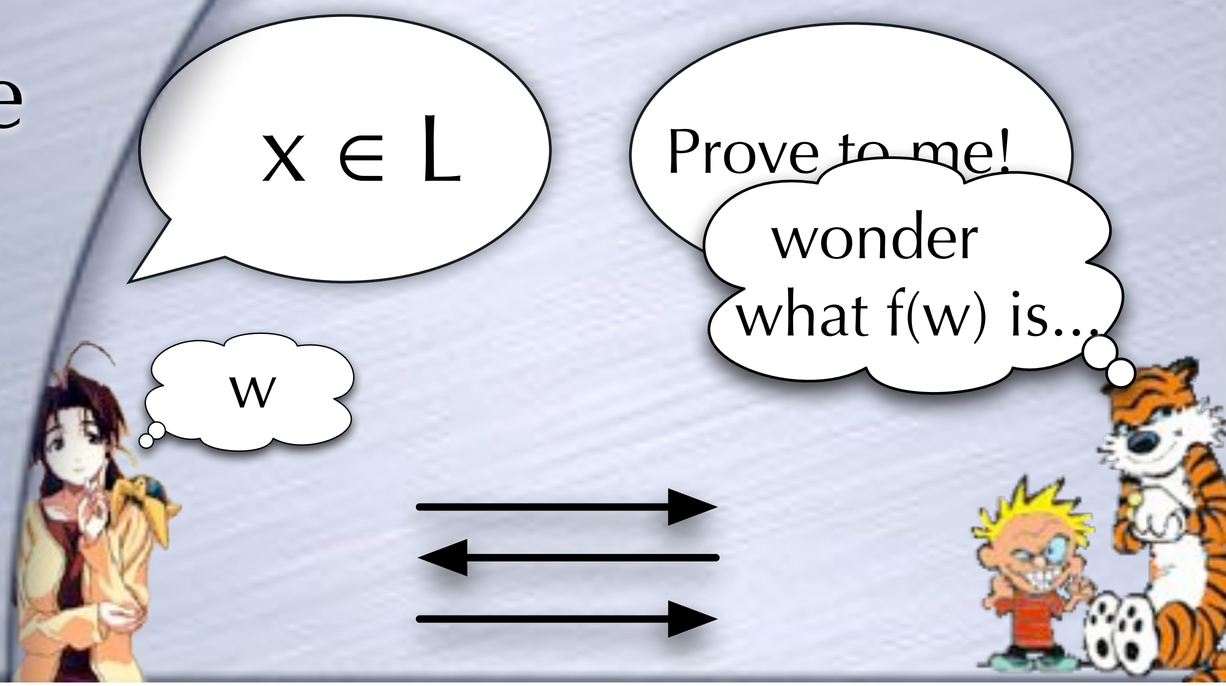
Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L



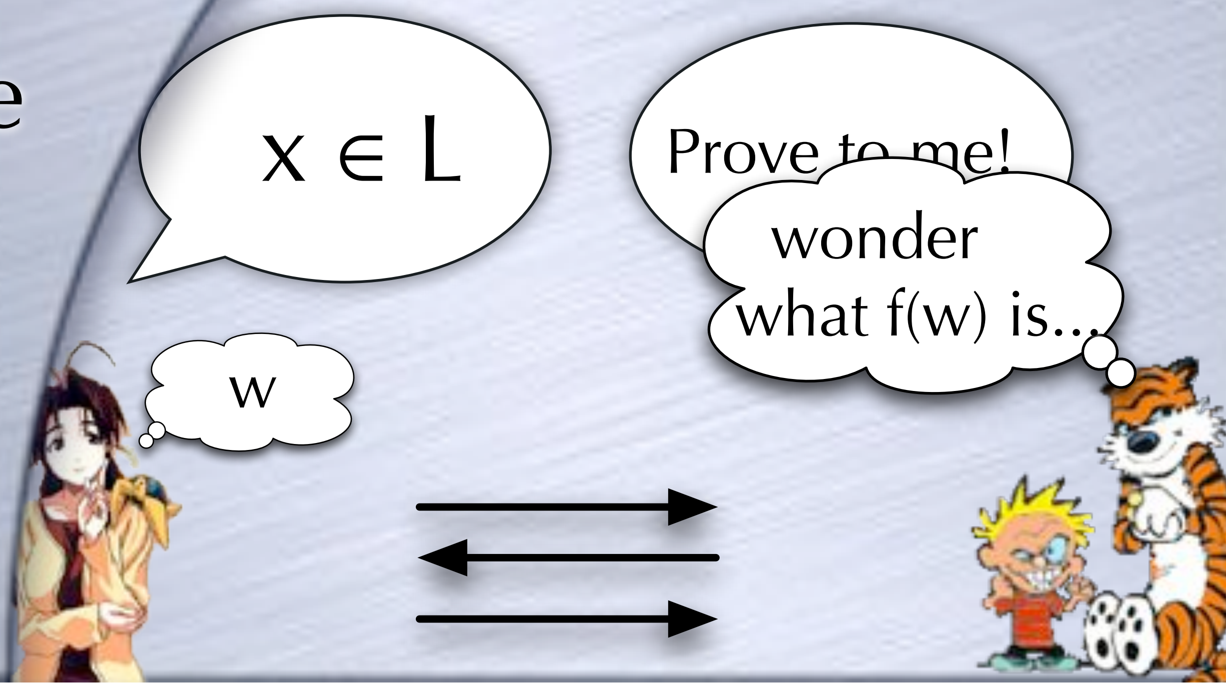
Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L
- How to formalize this?



Zero-Knowledge Proofs

- Verifier should not gain ***any*** knowledge from the honest prover
  - except whether x is in L
- How to formalize this?
  - Simulation!



An Example

An Example

- **Graph Isomorphism**

An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$

An Example

- **Graph Isomorphism**
  - (G_0, G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ

An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?

An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?

$G^* := \pi(G_1)$
(random π)

An Example

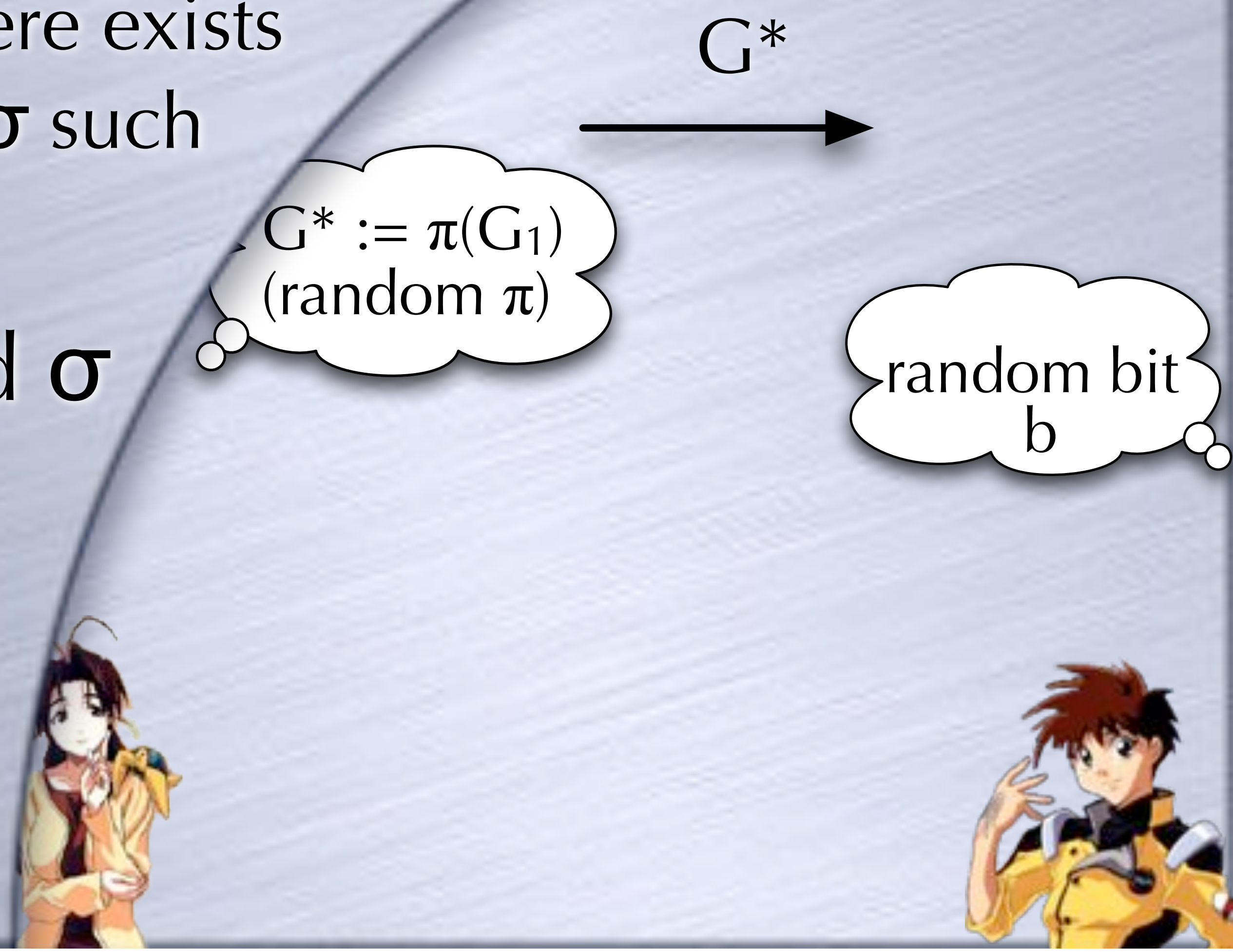
- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?

$G^* := \pi(G_1)$
(random π)

G^* →

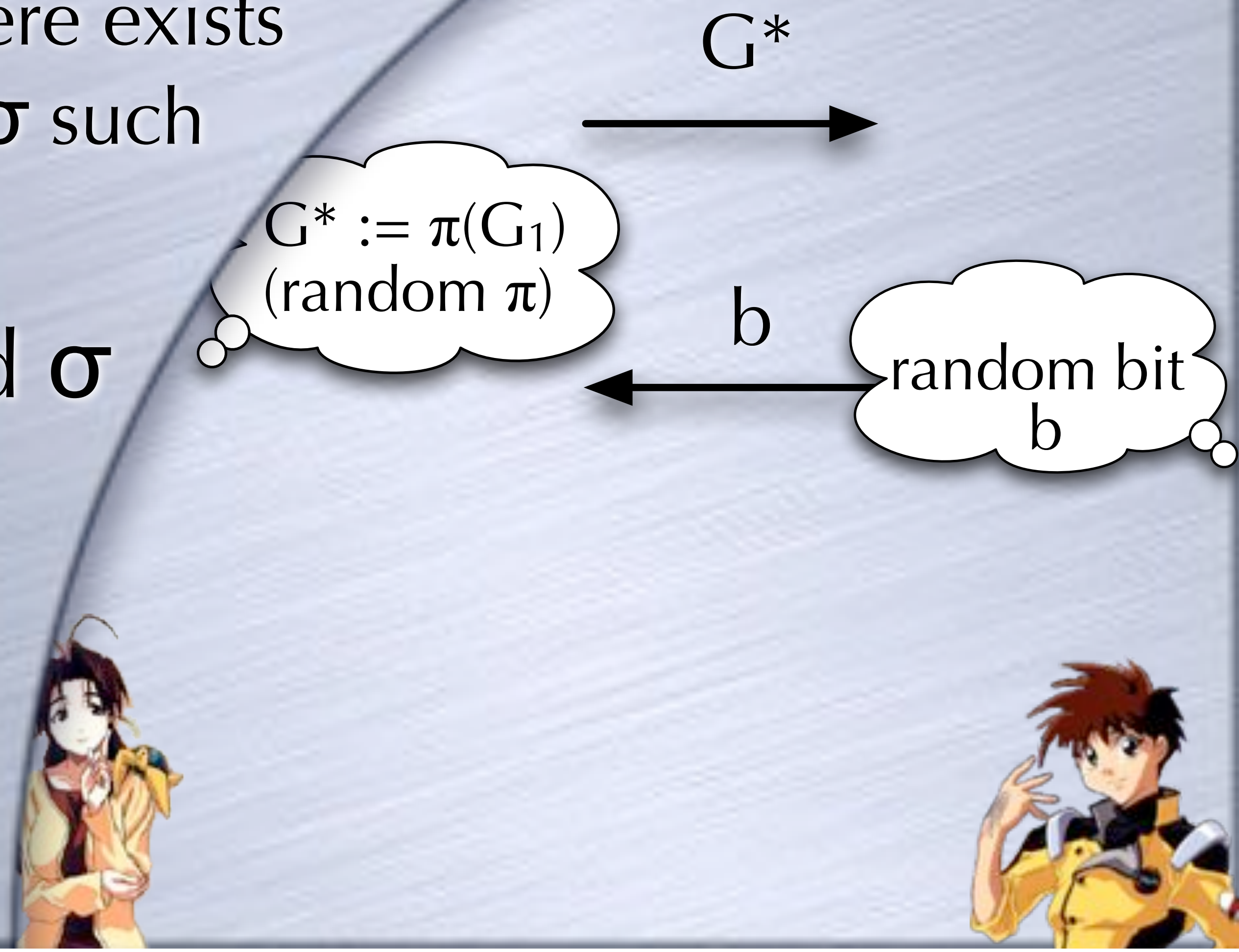
An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?



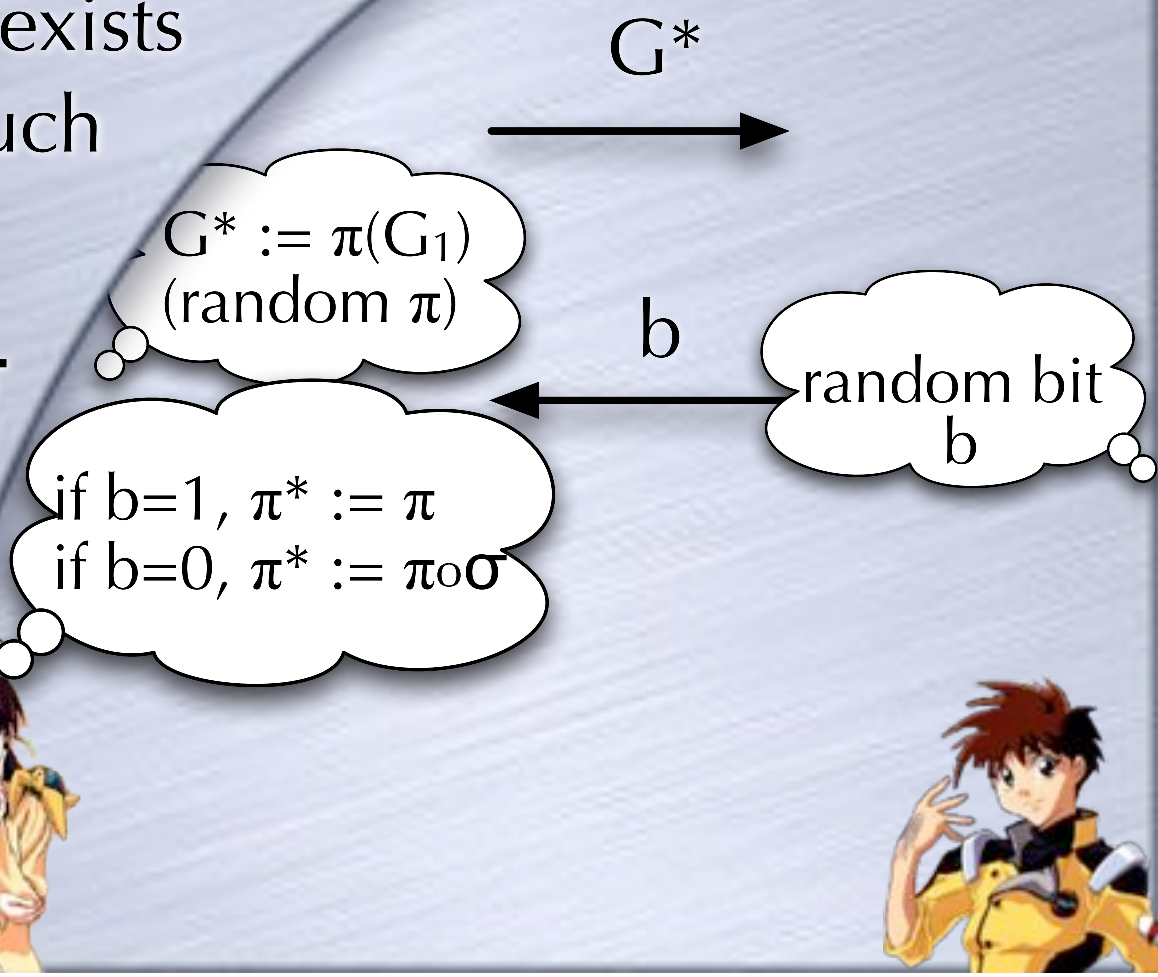
An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?



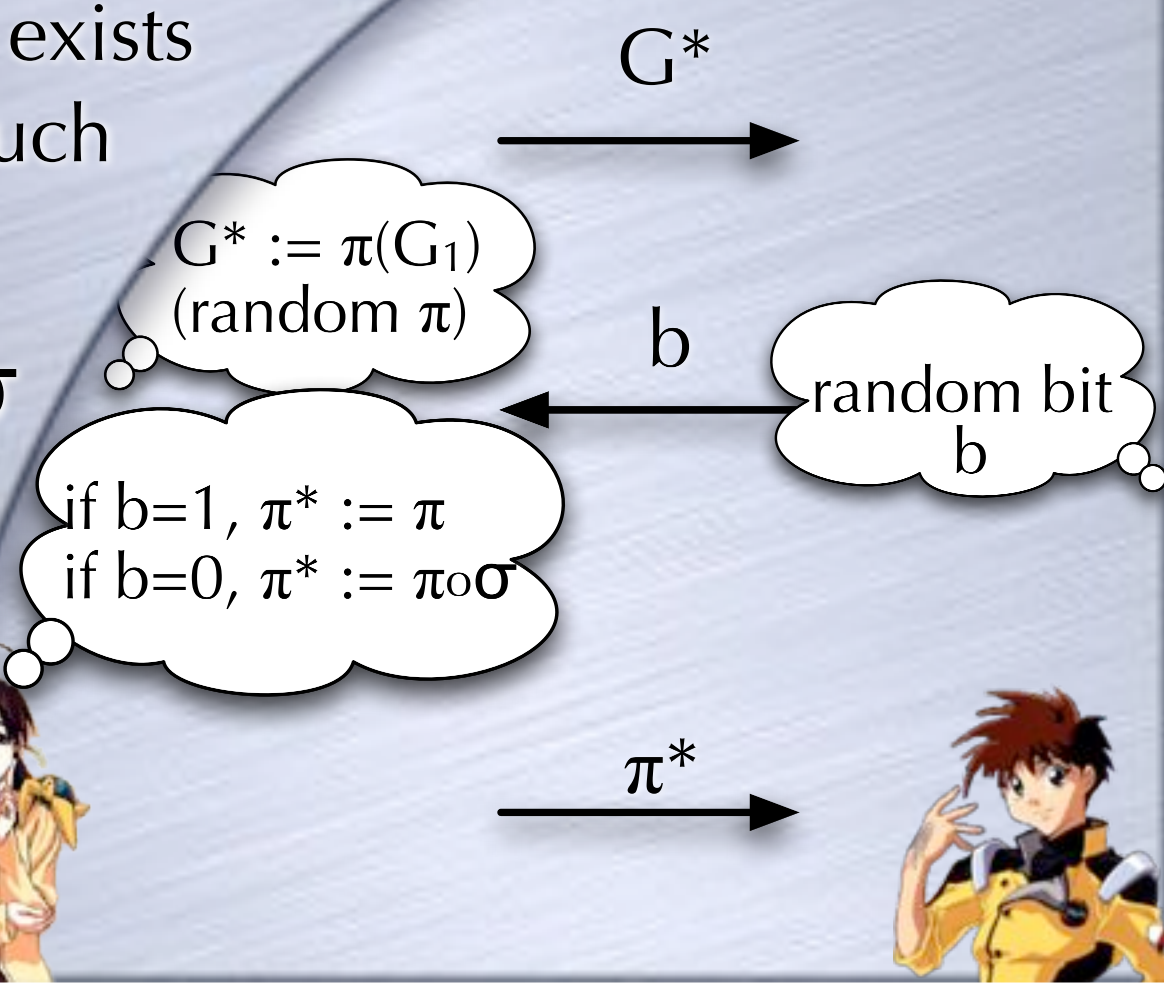
An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?



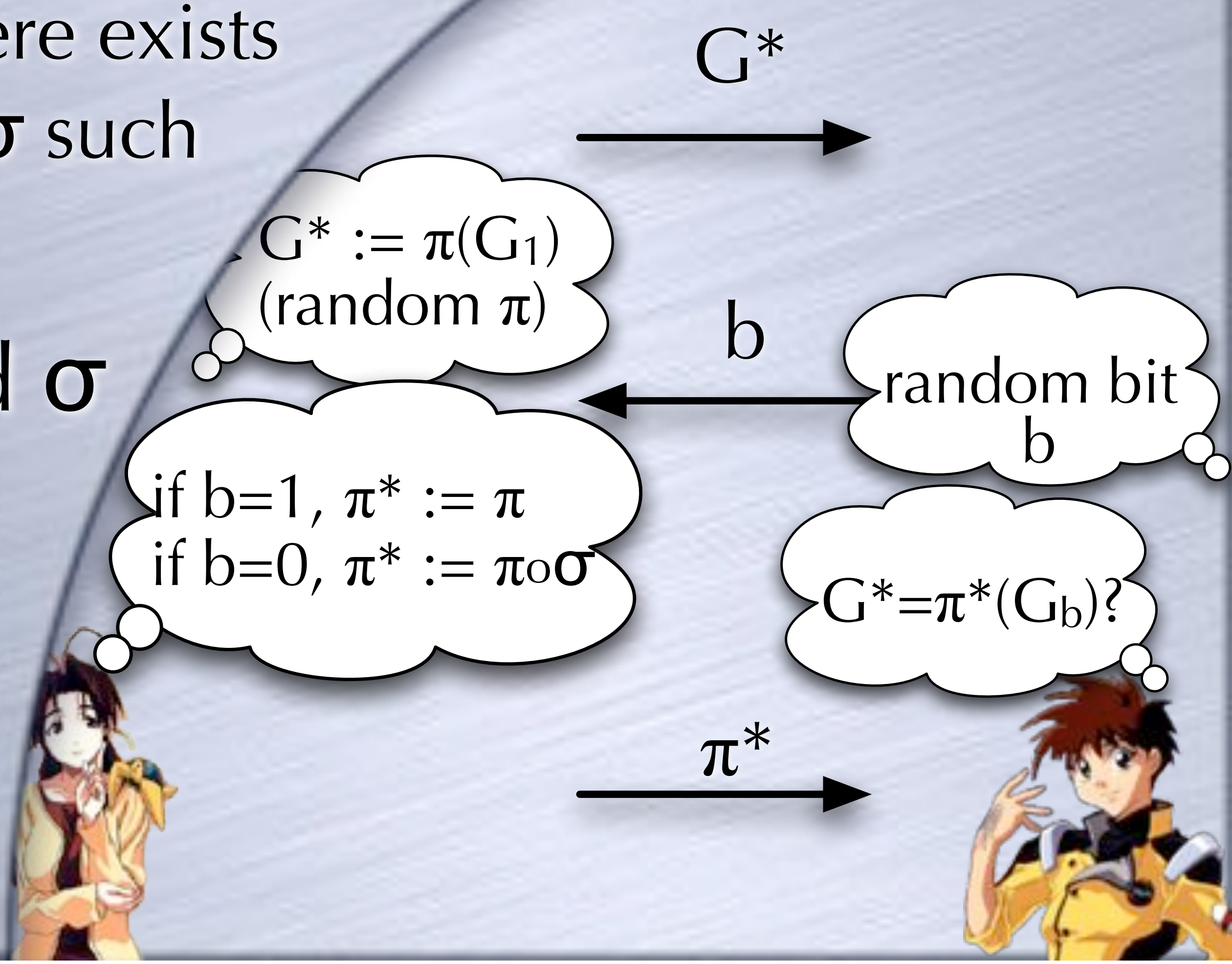
An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?

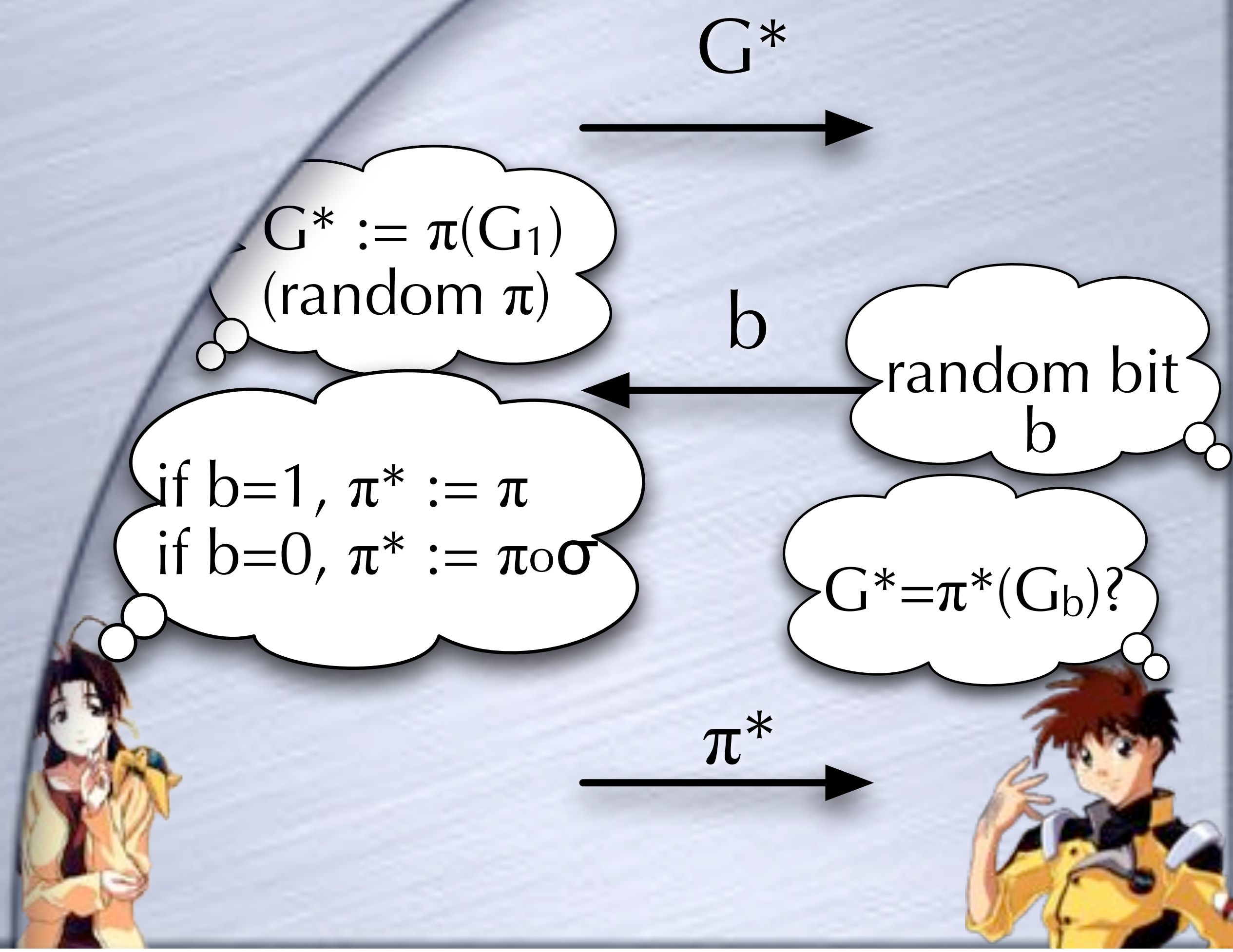


An Example

- **Graph Isomorphism**
  - (G_0,G_1) in L iff there exists an isomorphism σ such that $\sigma(G_0)=G_1$
- IP protocol: send σ
- ZK protocol?

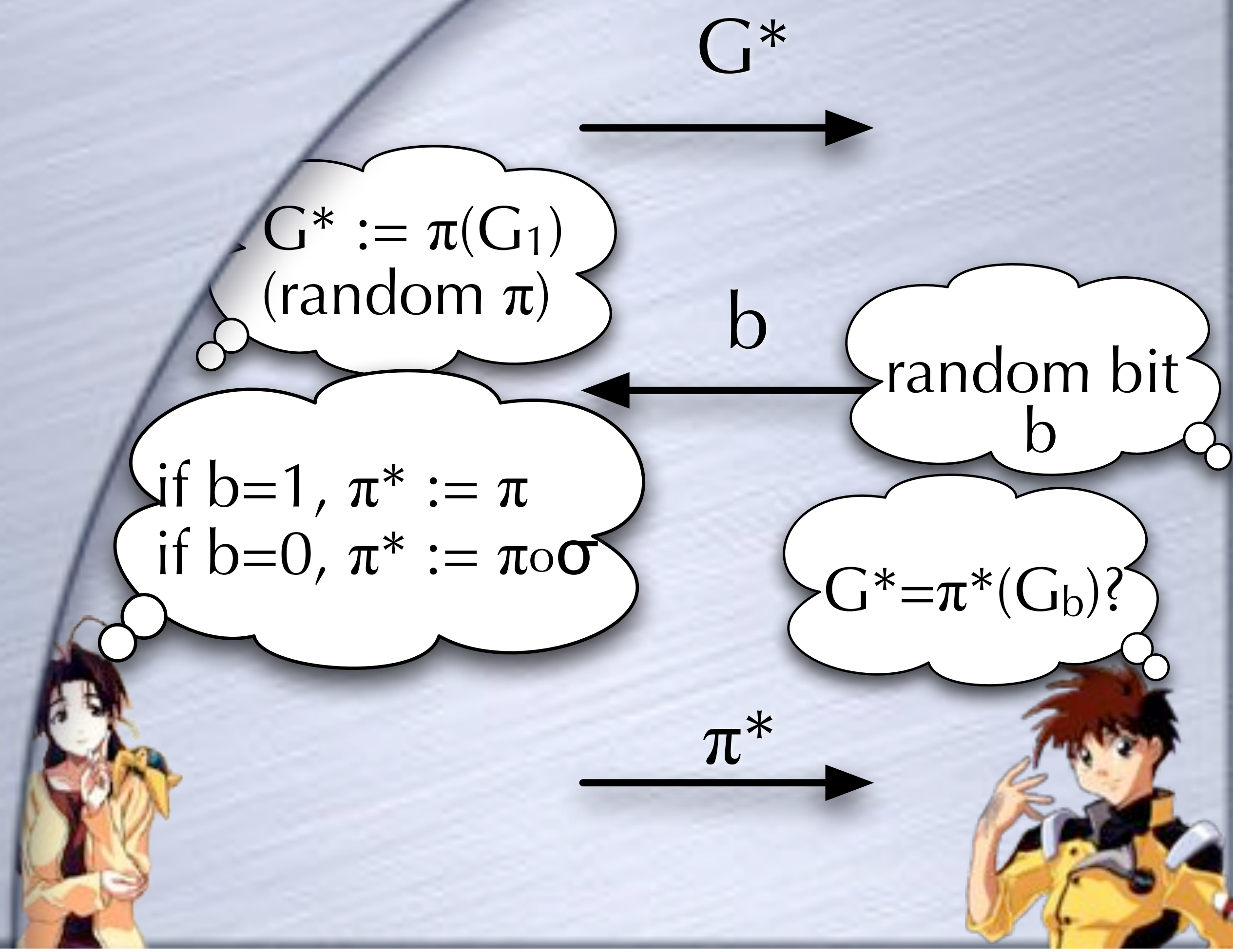


An Example



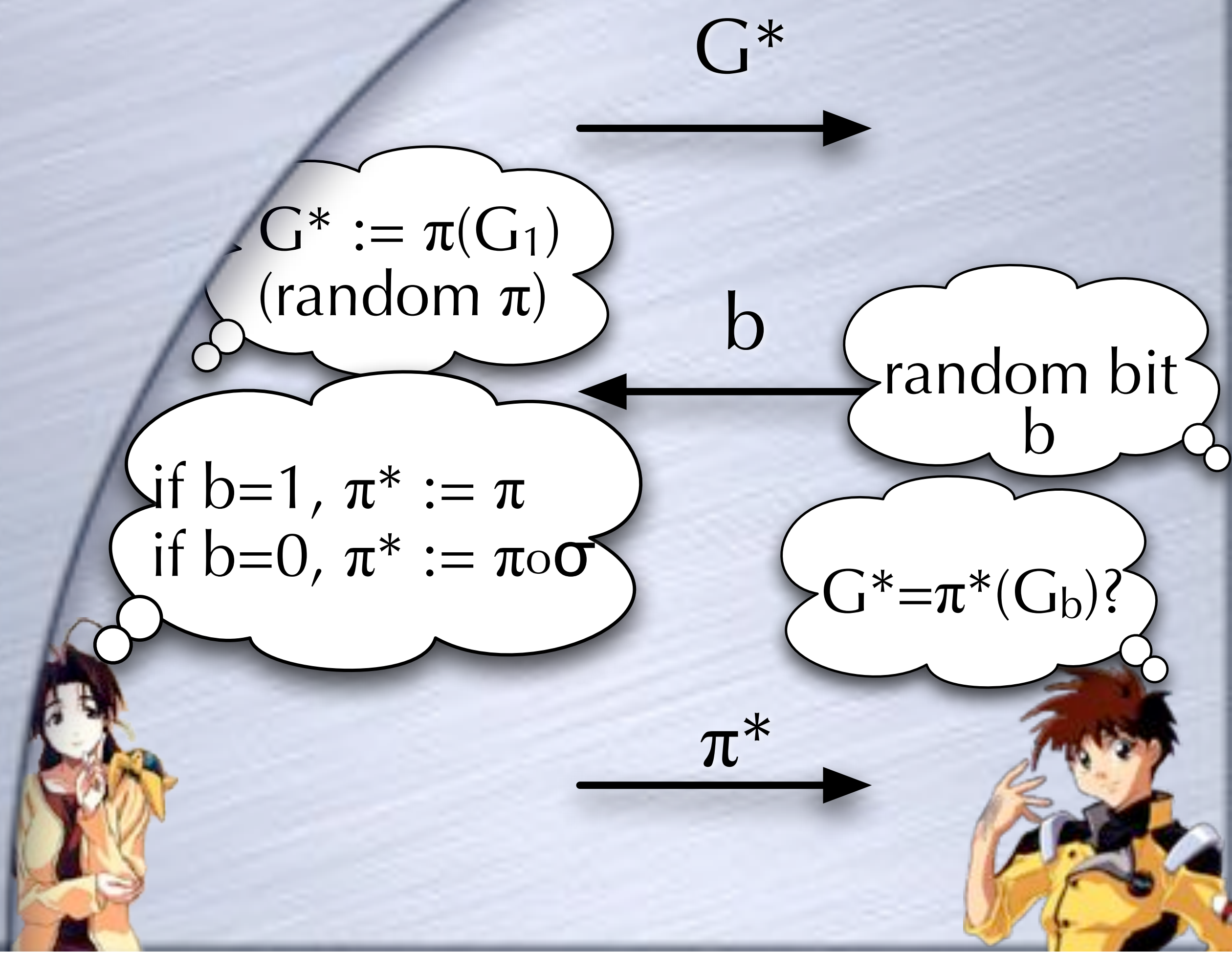
An Example

- **Why is this convincing?**



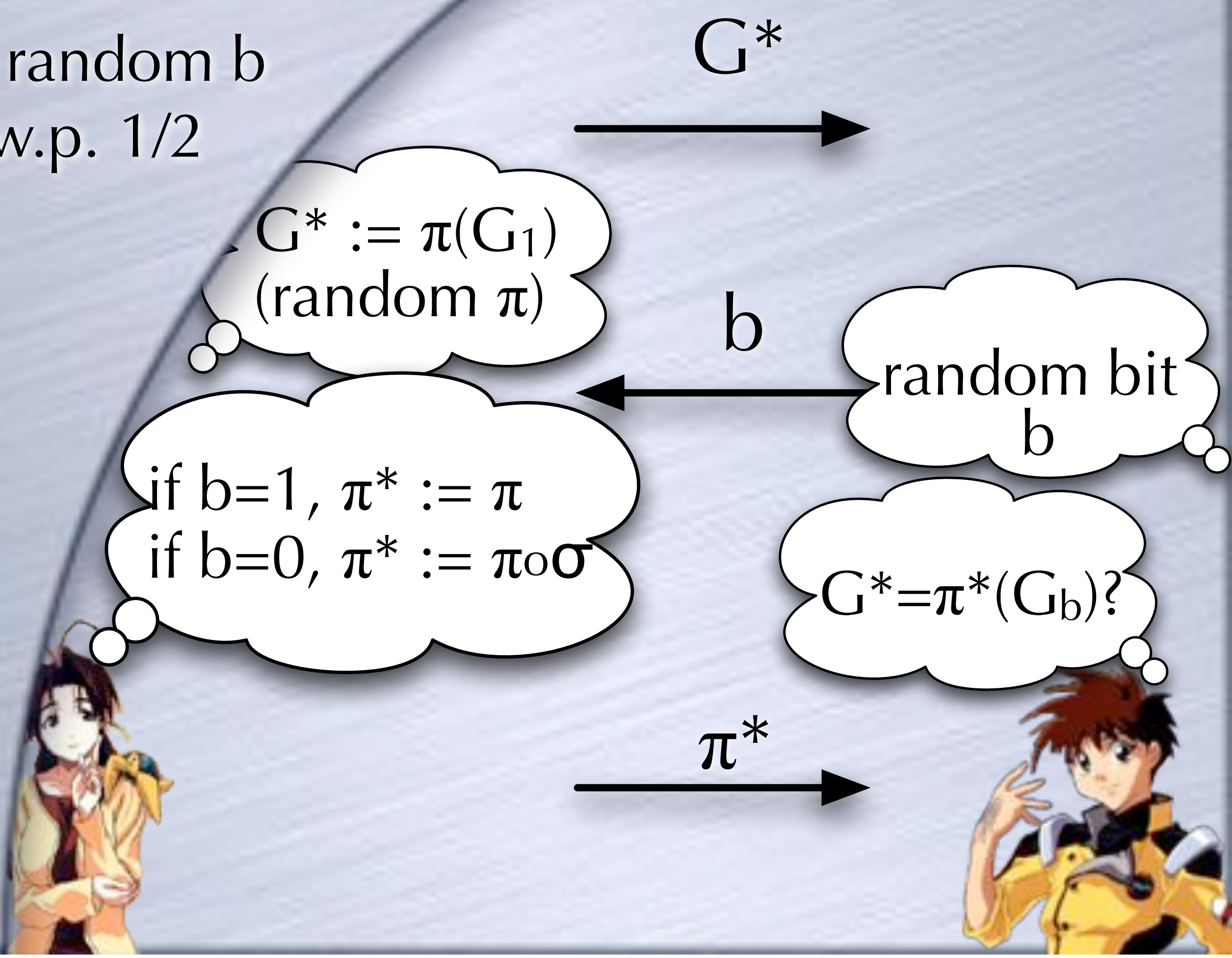
An Example

- **Why is this convincing?**
  - If prover can answer both b's for the same G* then $G_0 \sim G_1$



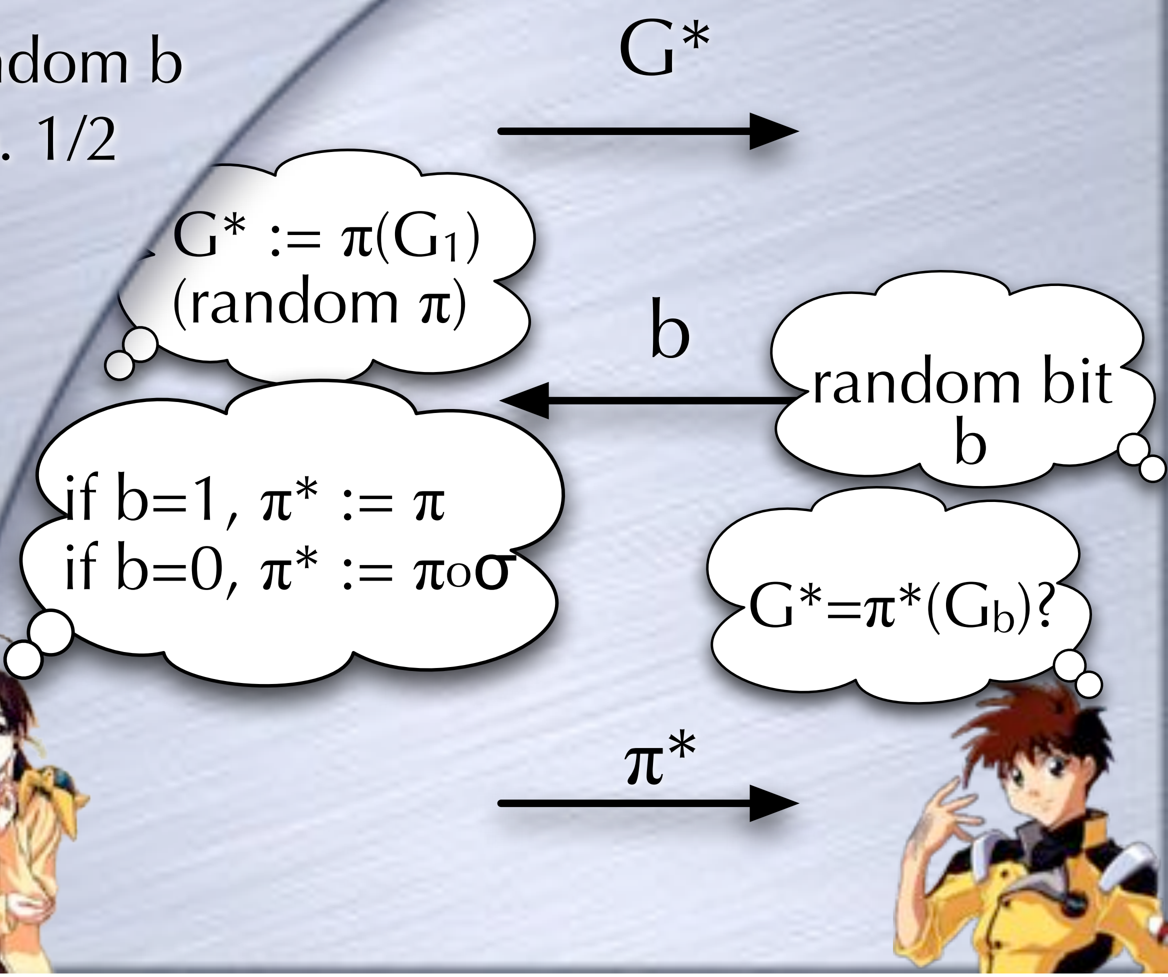
An Example

- **Why is this convincing?**
  - If prover can answer both b's for the same G* then $G_0 \sim G_1$
  - Otherwise, testing on a random b will leave prover stuck w.p. 1/2



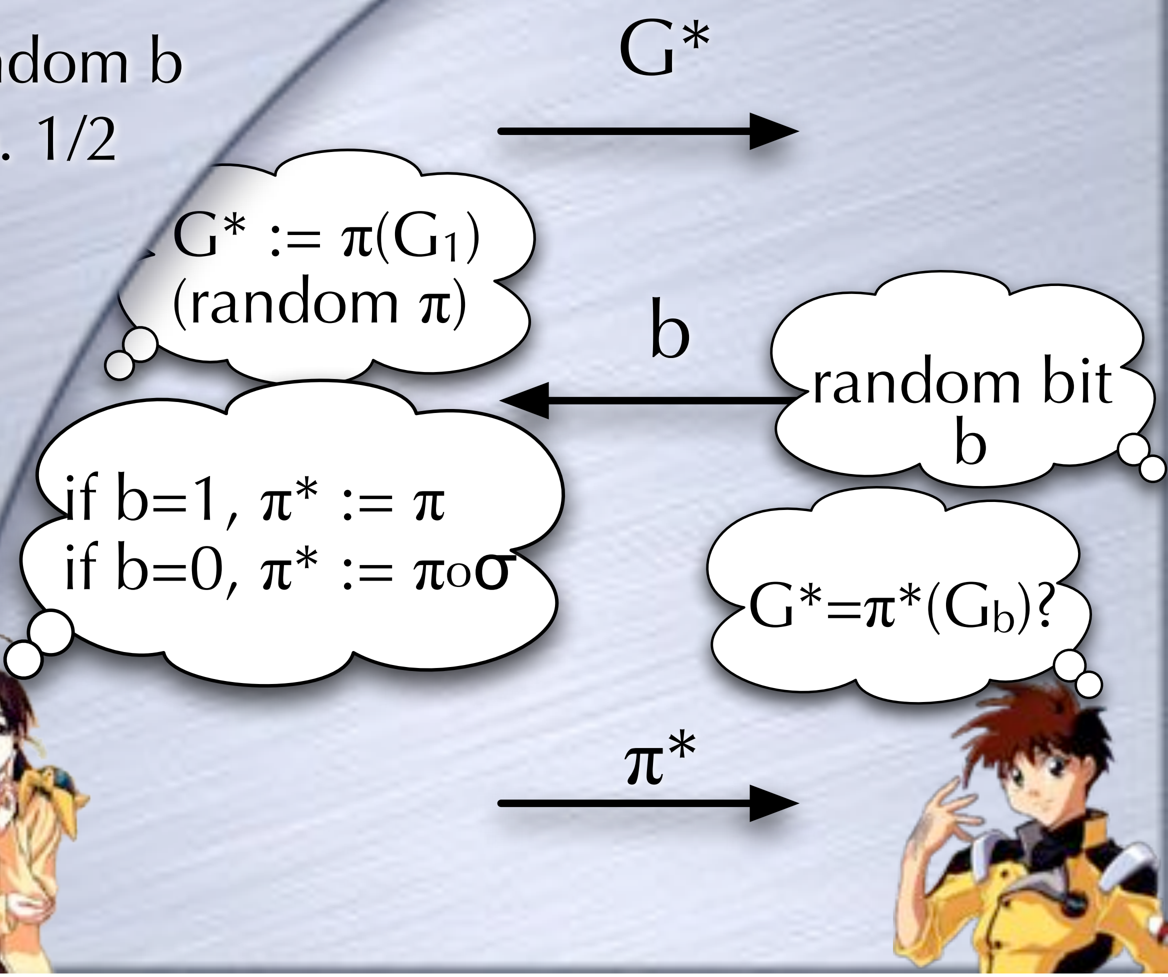
An Example

- **Why is this convincing?**
  - If prover can answer both b's for the same G^* then $G_0 \sim G_1$
  - Otherwise, testing on a random b will leave prover stuck w.p. 1/2
- **Why ZK?**



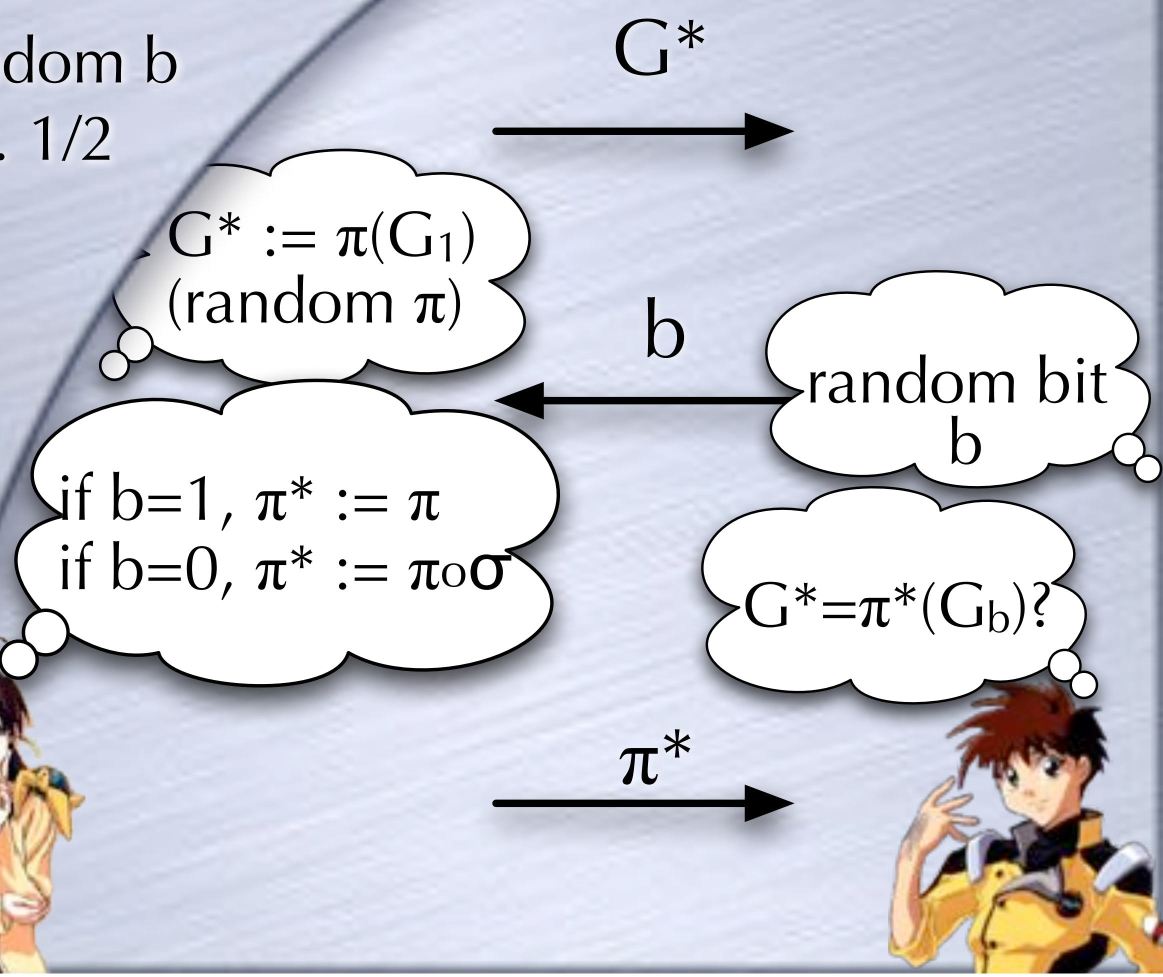
An Example

- **Why is this convincing?**
  - If prover can answer both b's for the same G* then $G_0 \sim G_1$
  - Otherwise, testing on a random b will leave prover stuck w.p. 1/2
- **Why ZK?**
  - Verifier's view: random b and π^* s.t. $G^*=\pi^*(G_b)$



An Example

- **Why is this convincing?**
  - If prover can answer both b's for the same G^* then $G_0 \sim G_1$
  - Otherwise, testing on a random b will leave prover stuck w.p. 1/2
- **Why ZK?**
  - Verifier's view: random b and π^* s.t. $G^* = \pi^*(G_b)$
  - Which he could have generated by himself (whether $G_0 \sim G_1$ or not)



Zero-Knowledge Proofs

Zero-Knowledge Proofs

- Interactive Proof

Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound

Zero-Knowledge Proofs

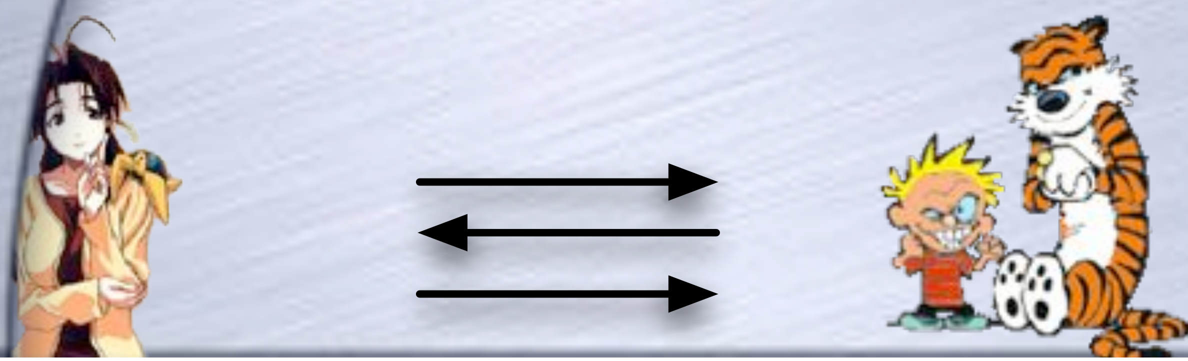
- Interactive Proof
  - Complete and Sound
- ZK Property:

Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:

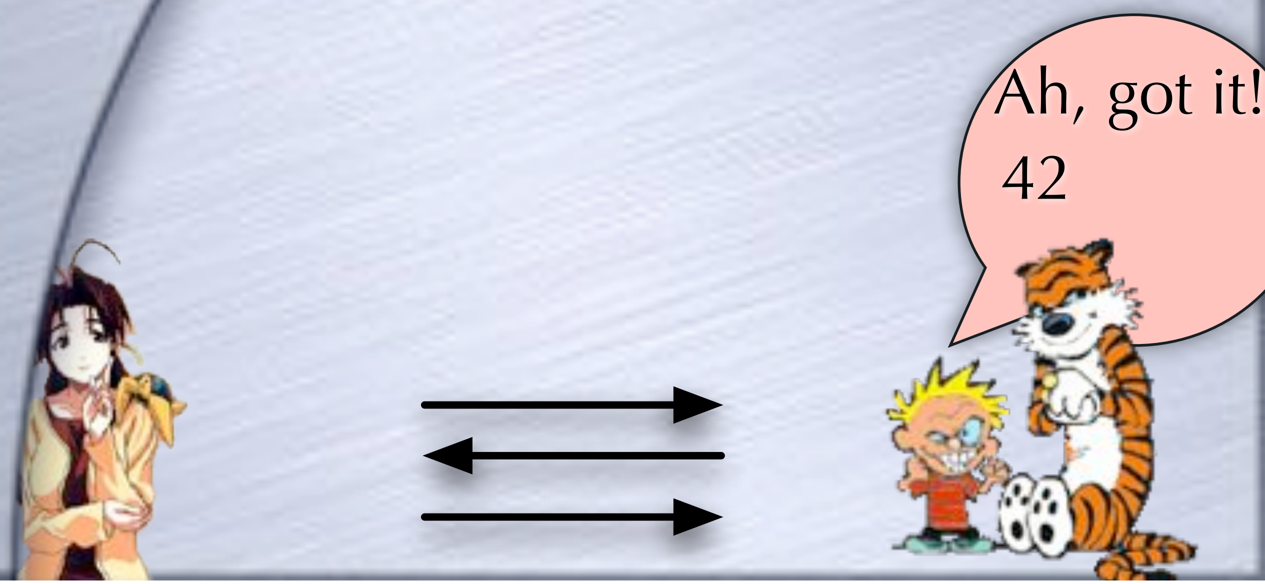
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:



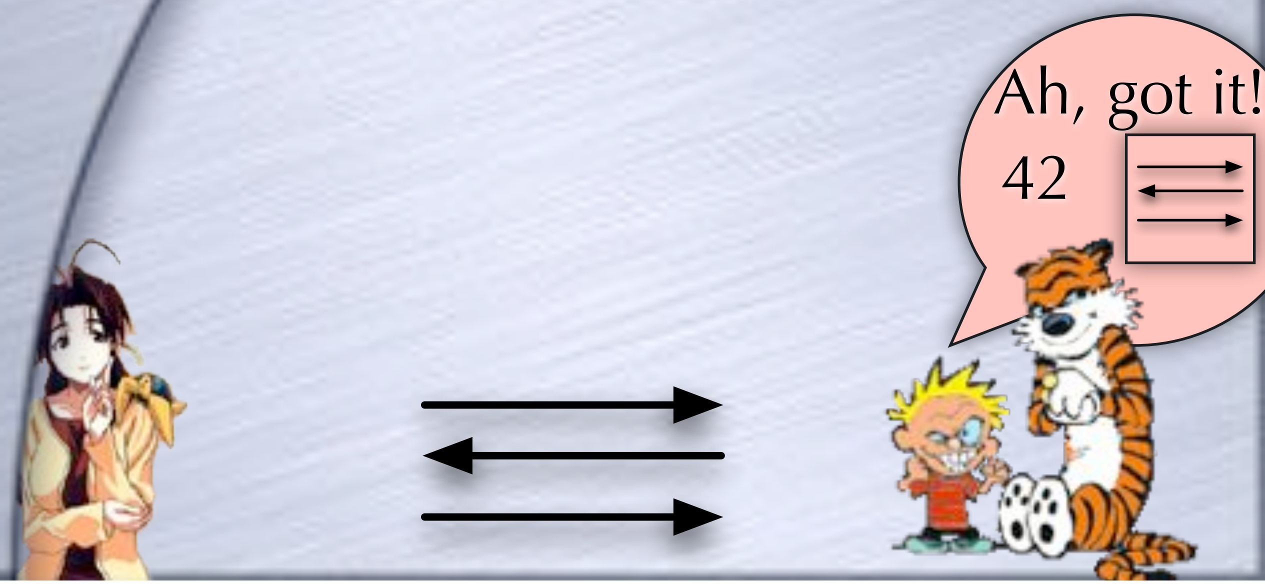
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:



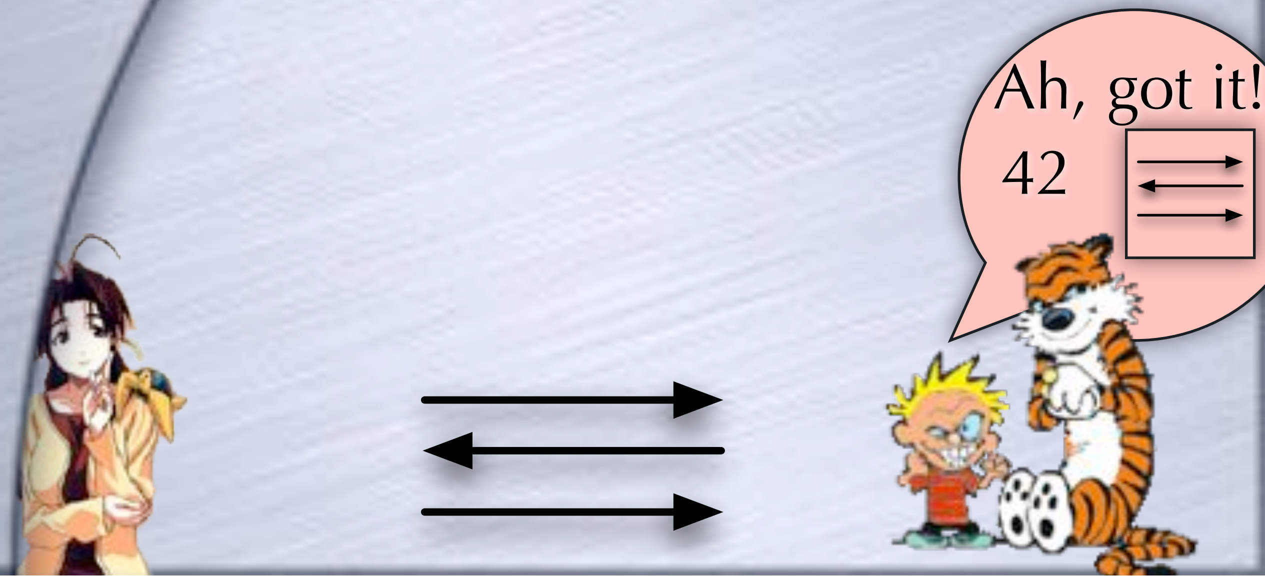
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:



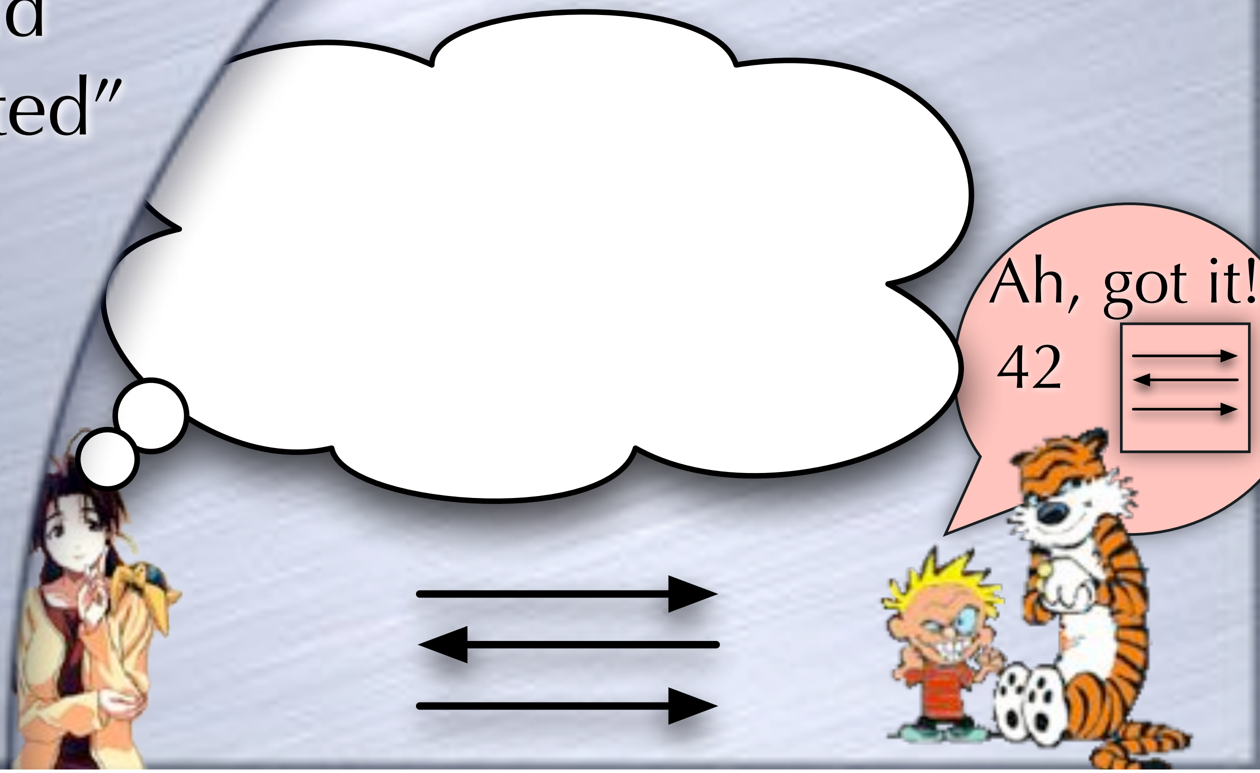
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"



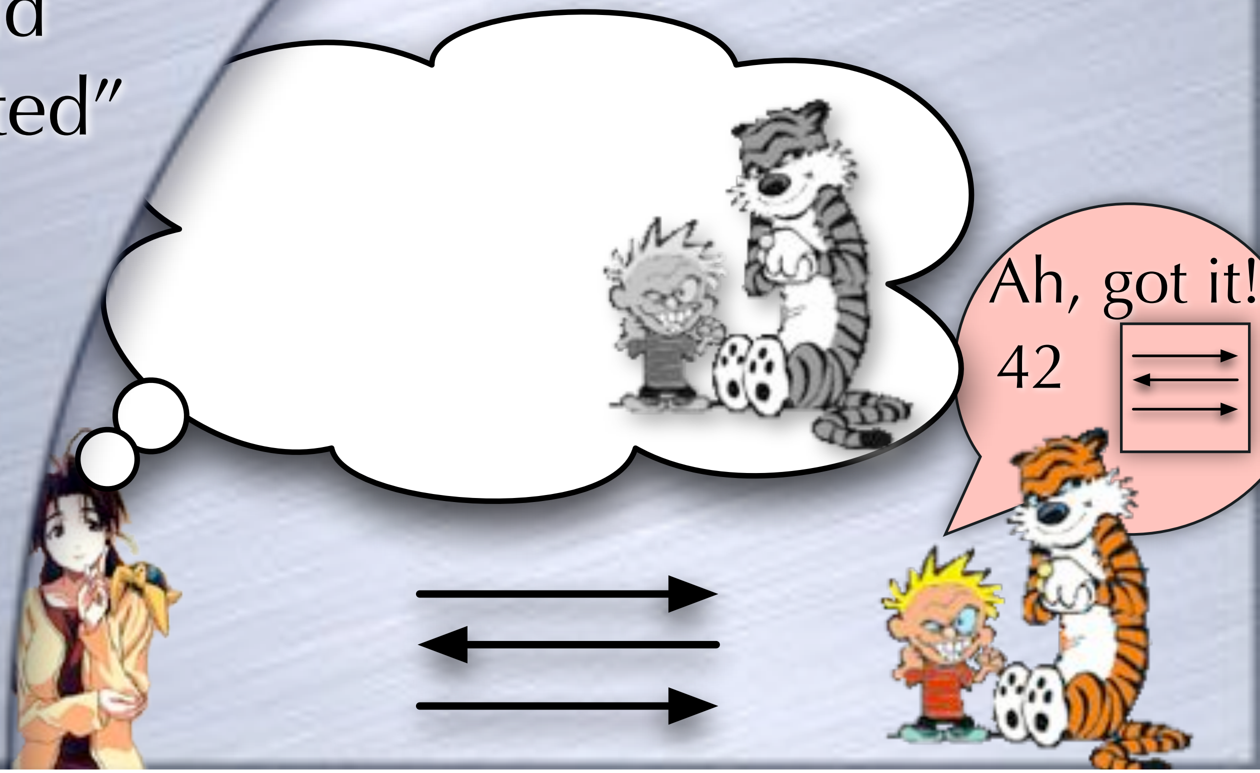
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"



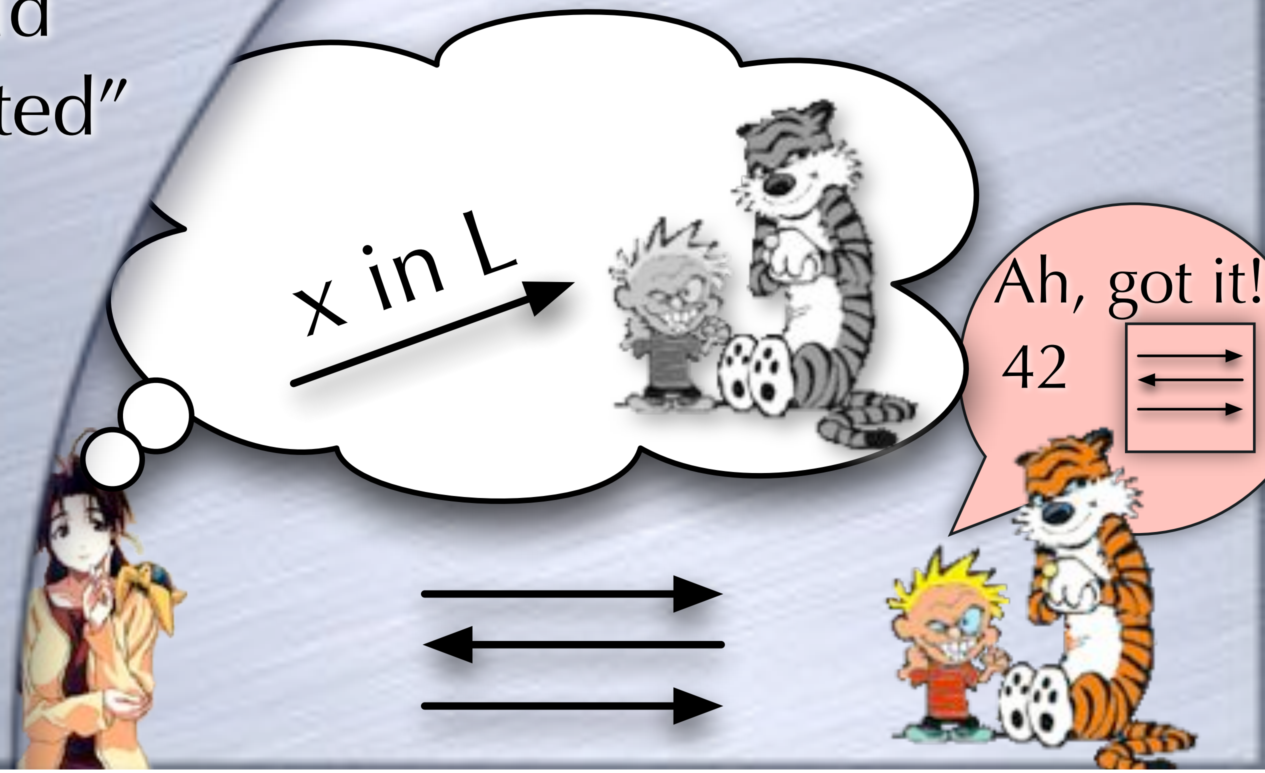
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"



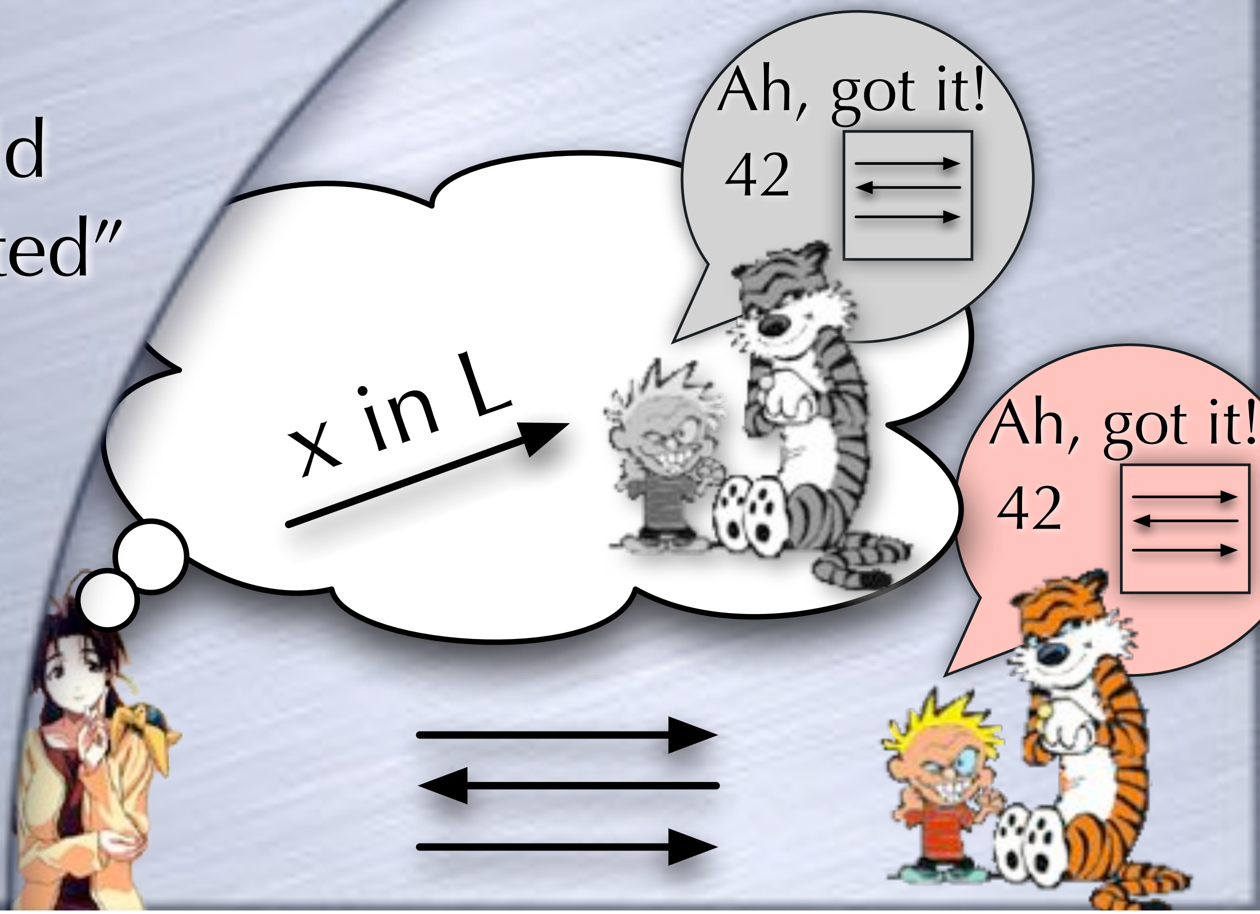
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"



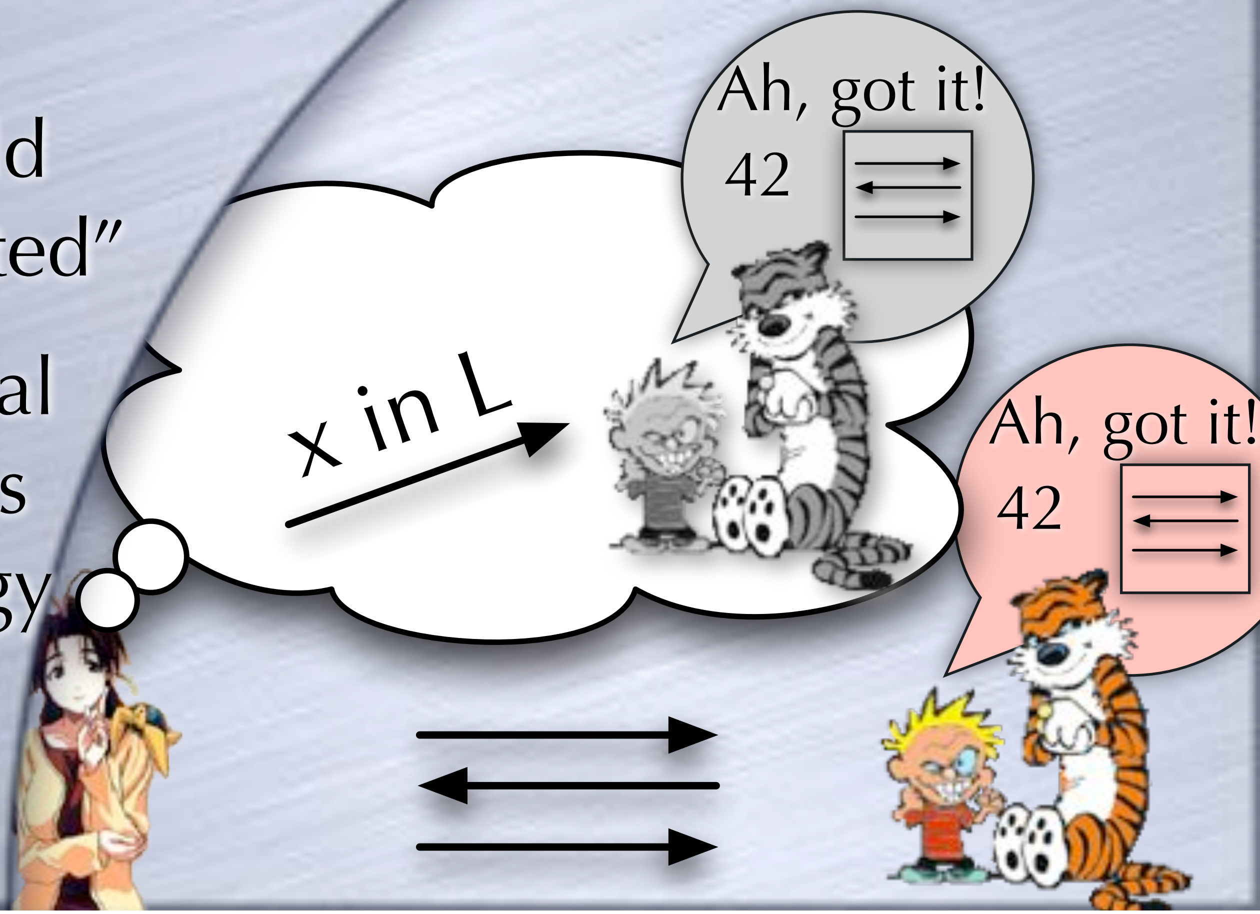
Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"

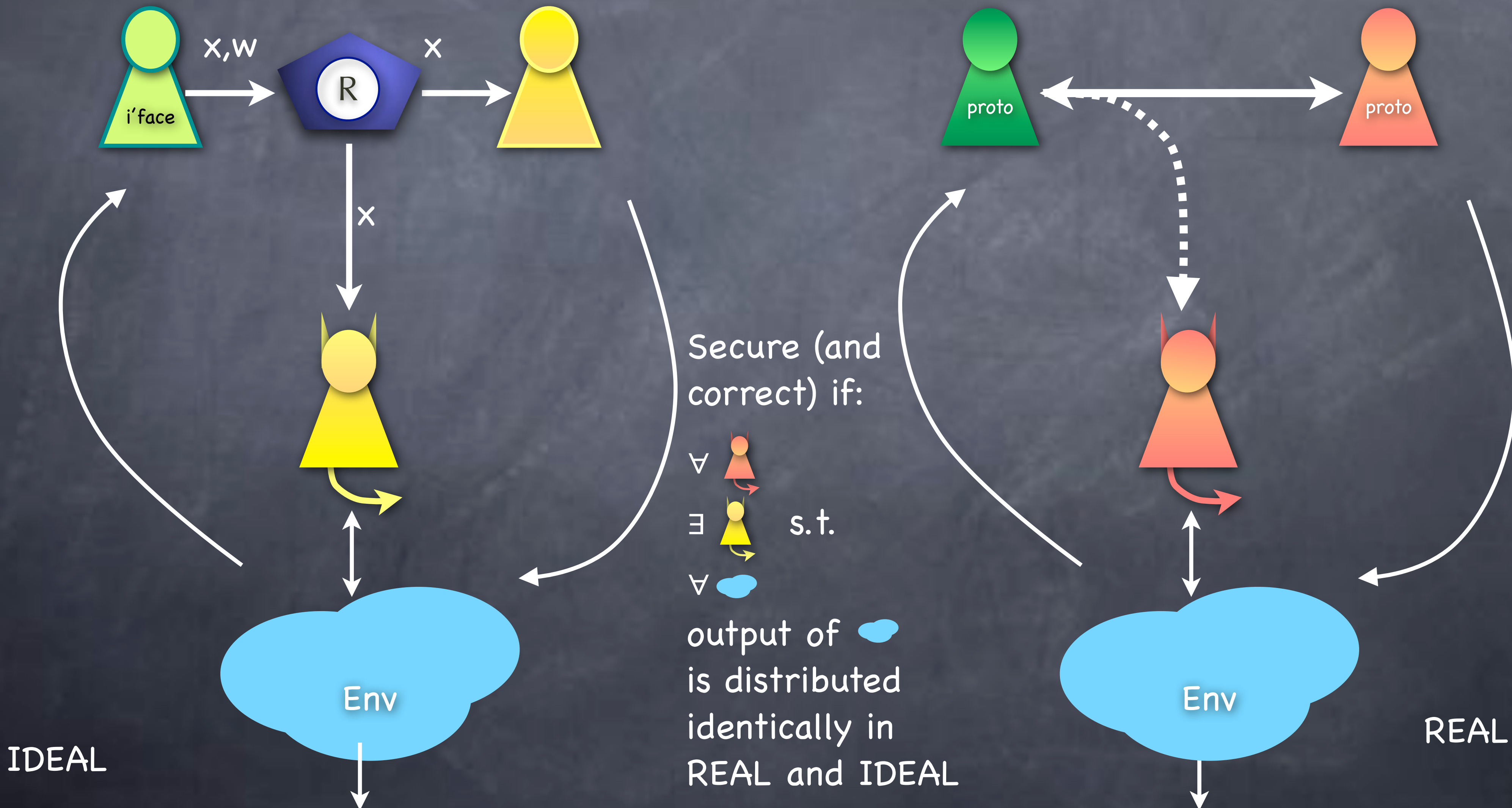


Zero-Knowledge Proofs

- Interactive Proof
  - Complete and Sound
- ZK Property:
  - Verifier's view could have been "simulated"
  - For every adversarial strategy, there exists a simulation strategy



ZK Property (in other pict's)



Secure (and correct) if:

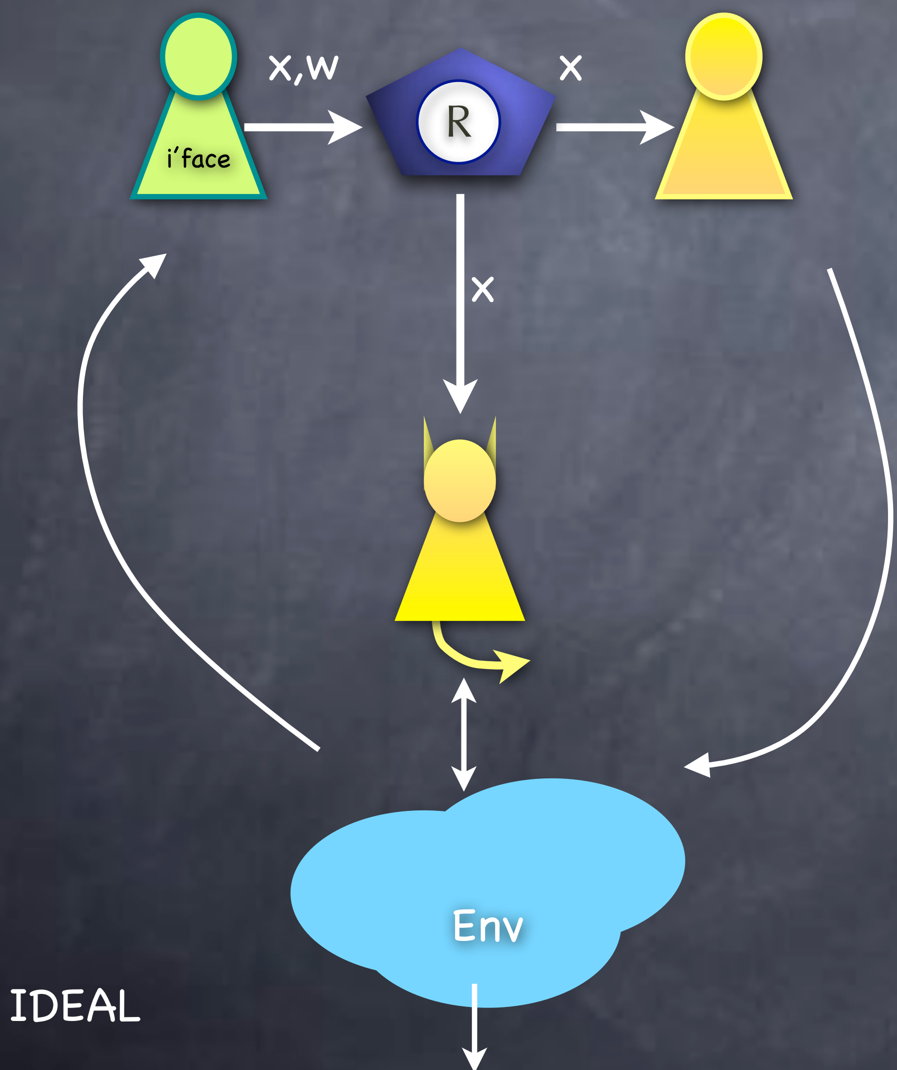
∀ [adversary in REAL]

∃ [adversary in IDEAL] s.t.

∀ [environment]

output of [environment] is distributed identically in REAL and IDEAL

ZK Property (in other pict's)



Secure (and correct) if:

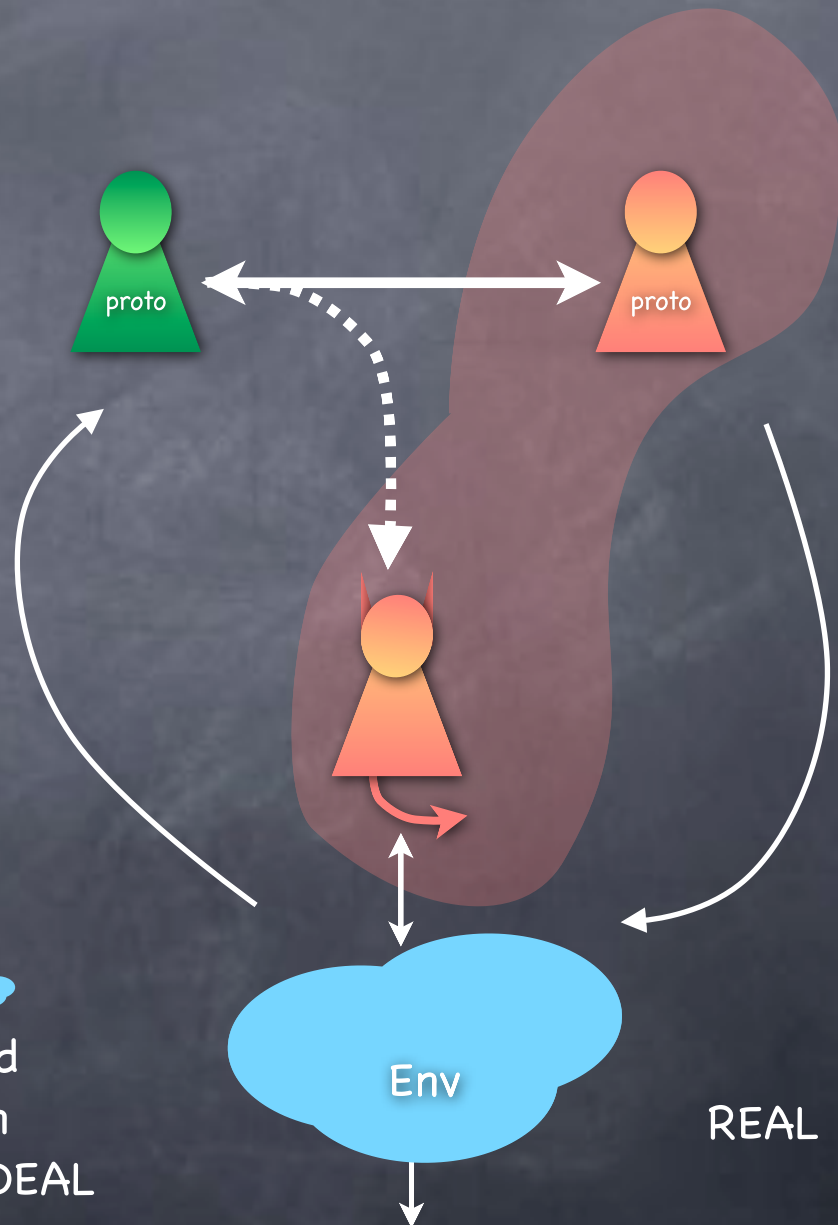
∀

∃ s.t.

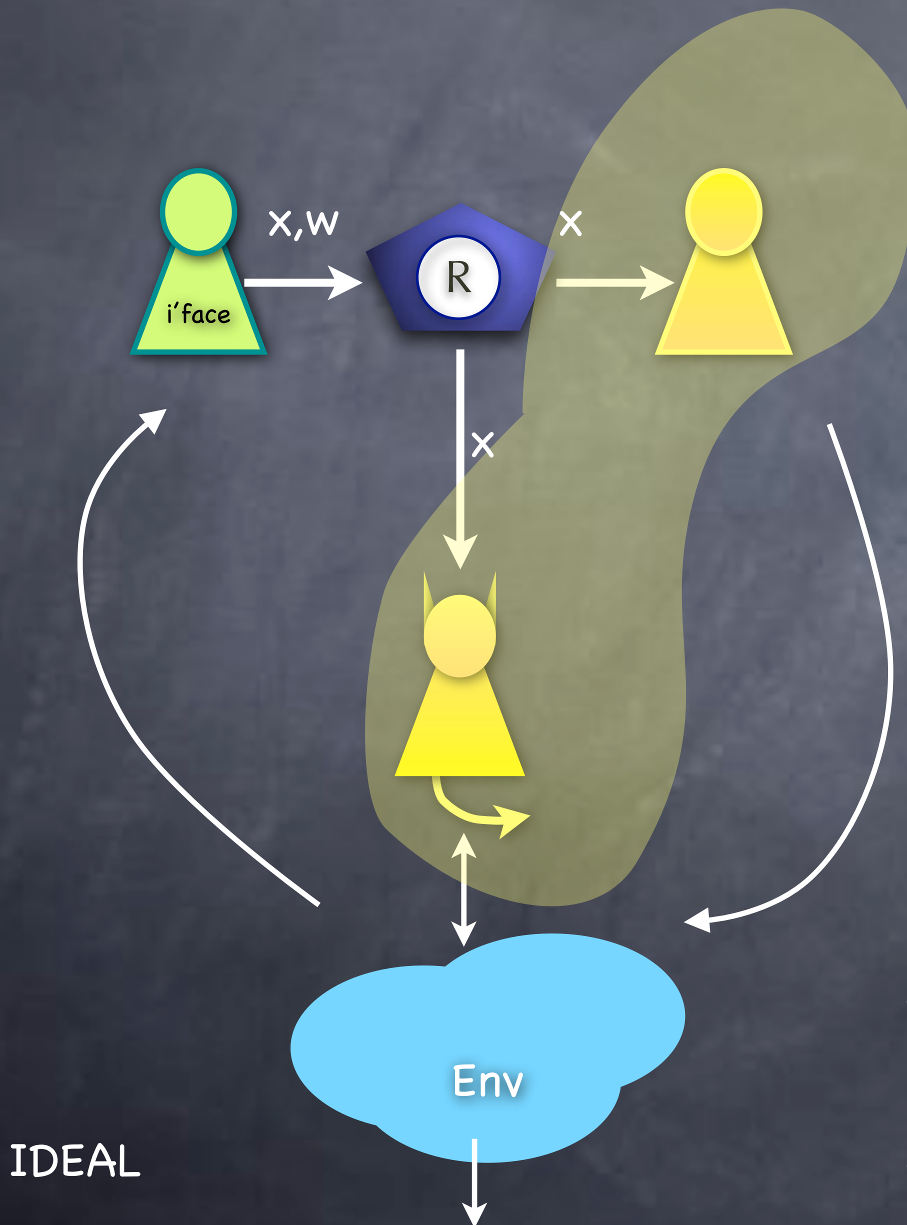
∀

output of

is distributed identically in REAL and IDEAL



ZK Property (in other pict's)



Secure (and correct) if:

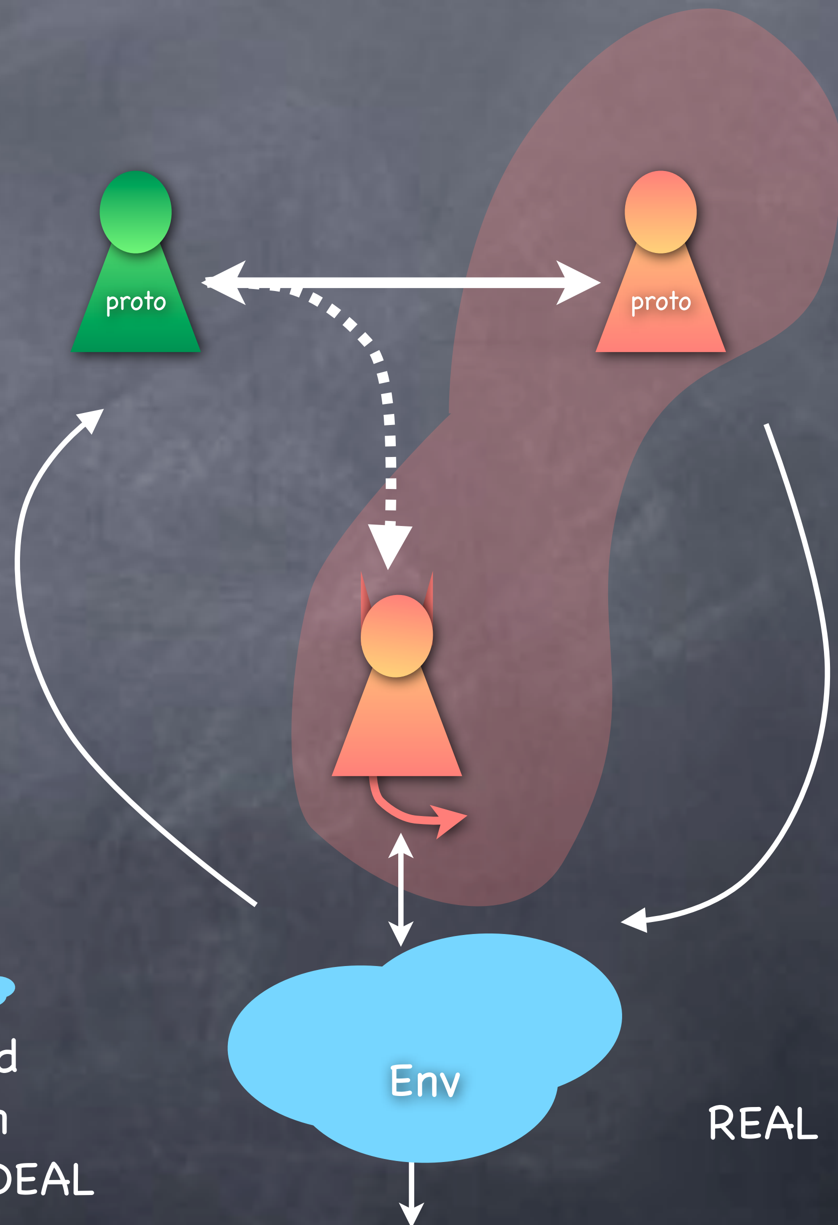
∀

∃ s.t.

∀

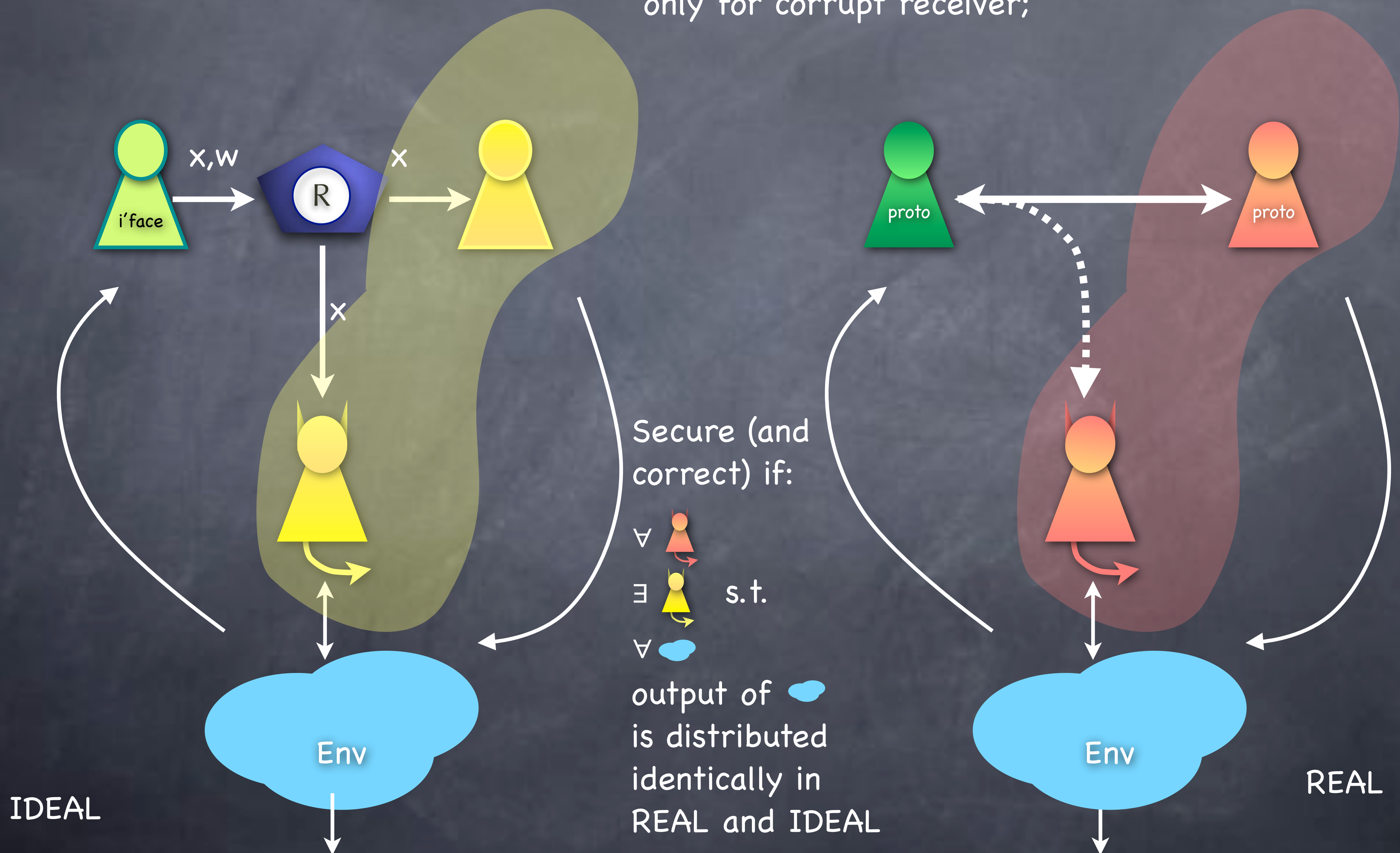
output of

is distributed identically in REAL and IDEAL



ZK Property (in other pict's)

Classical definition uses simulation only for corrupt receiver;



Secure (and correct) if:

∀ [corrupt party]

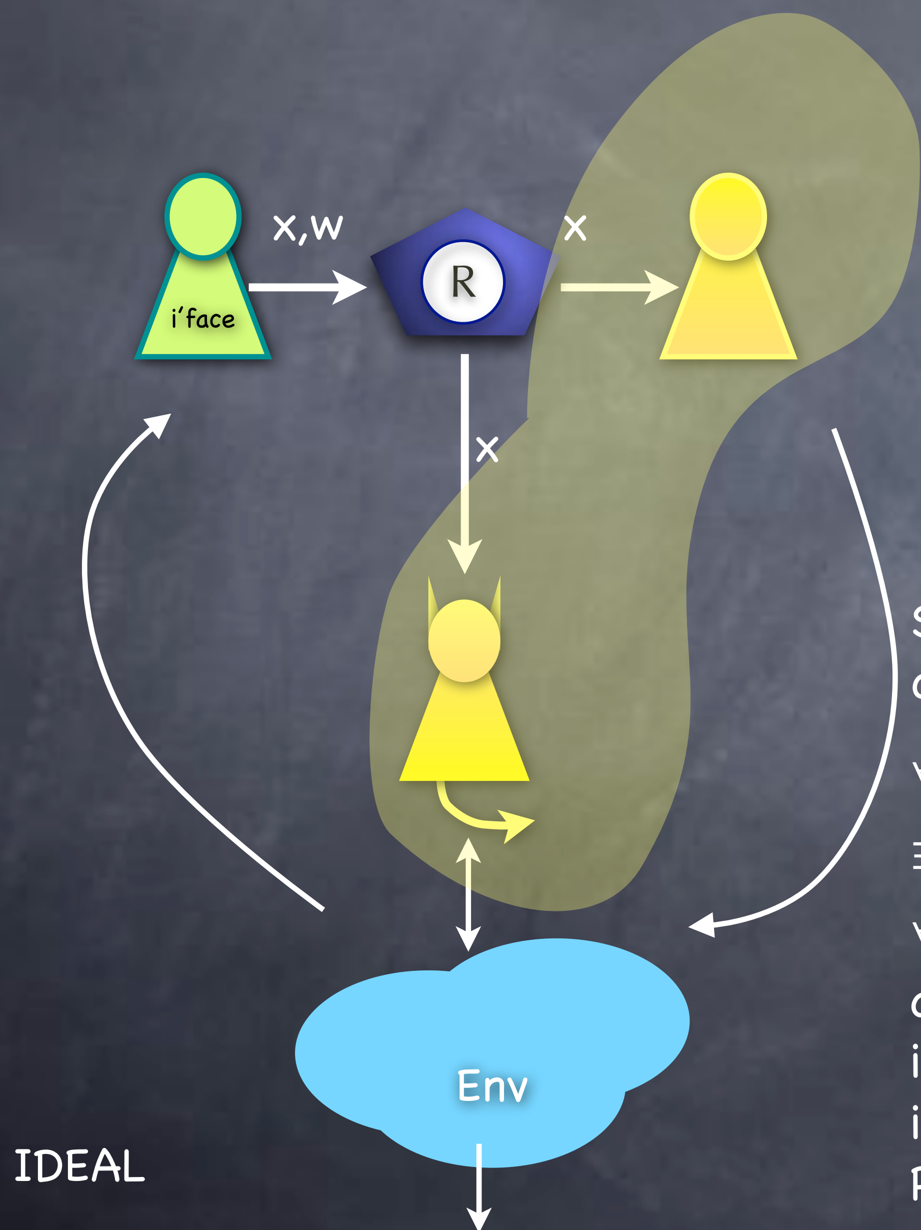
∃ [simulator] s.t.

∀ [environment]

output of [environment] is distributed identically in REAL and IDEAL

ZK Property (in other pict's)

Classical definition uses simulation only for corrupt receiver; and uses only standalone security: Environment gets only a transcript at the end



Secure (and correct) if:

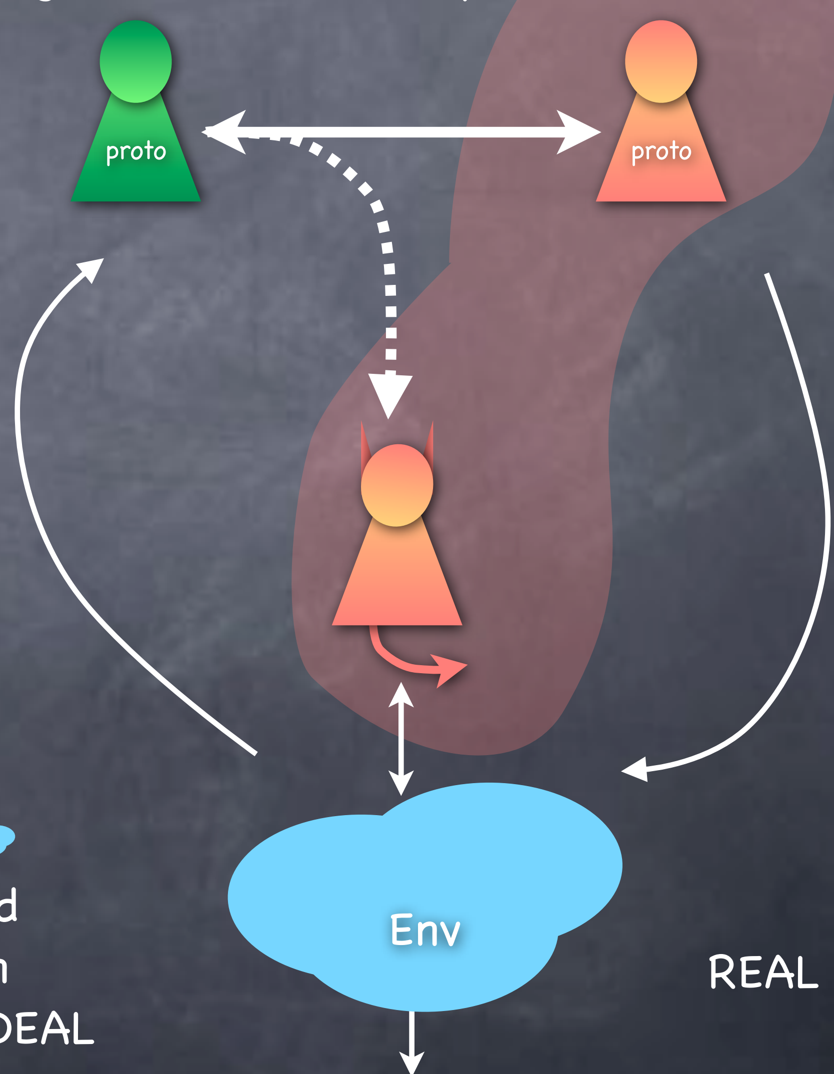
∀

∃ s.t.

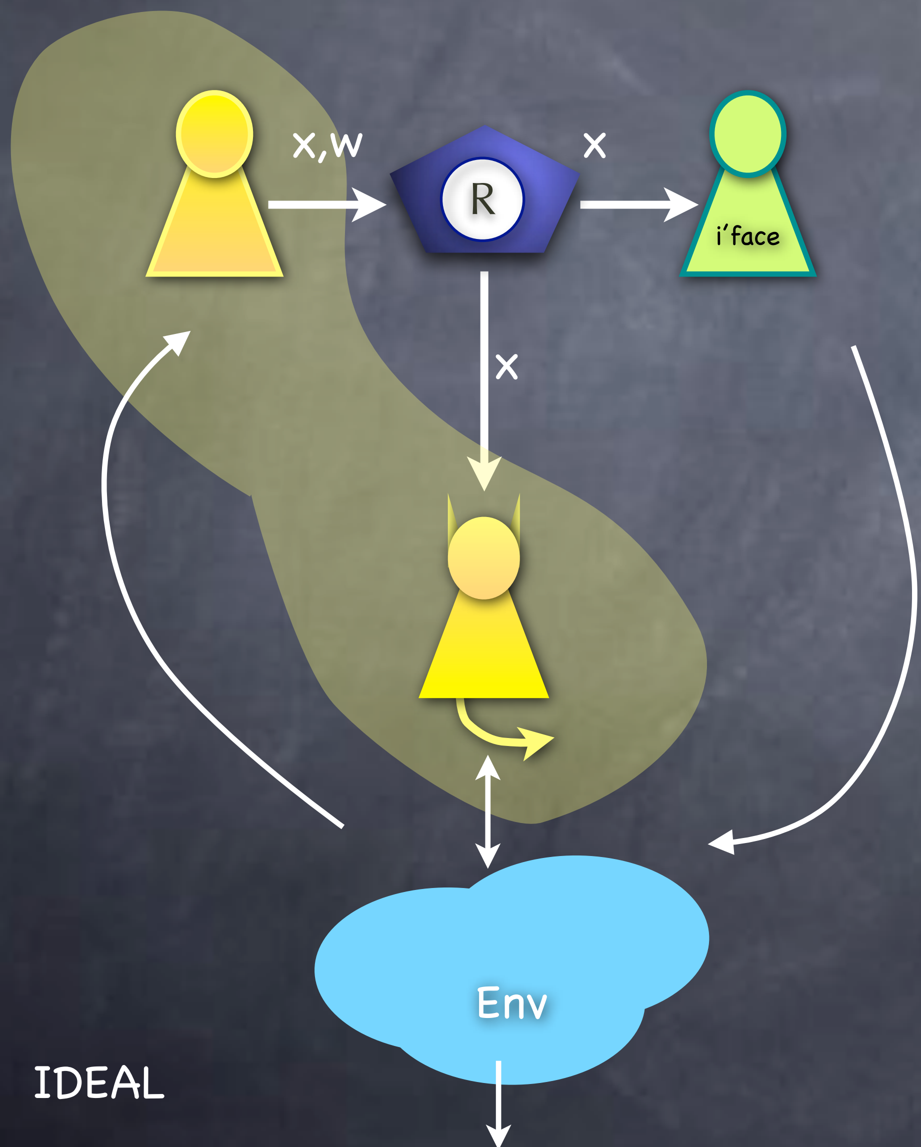
∀

output of

is distributed identically in REAL and IDEAL



SIM ZK



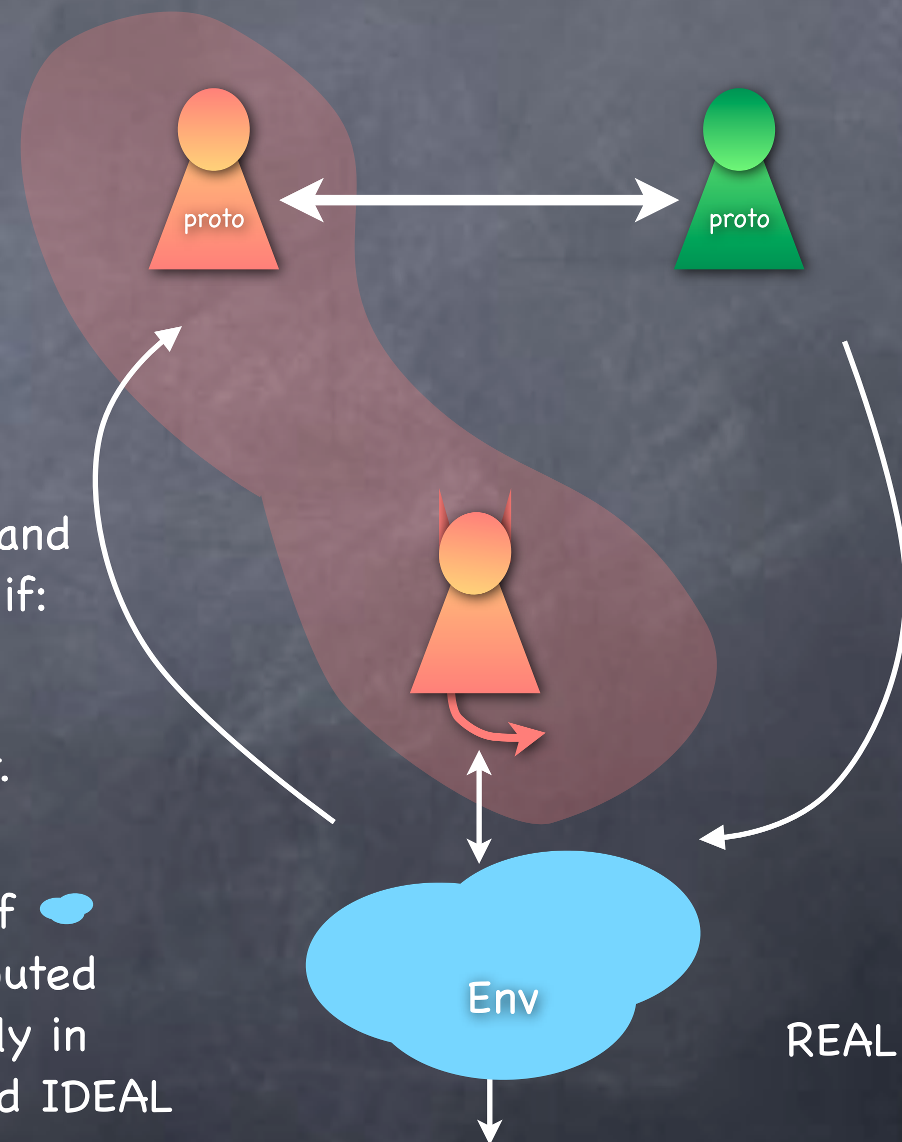
Secure (and correct) if:

∀

∃ s.t.

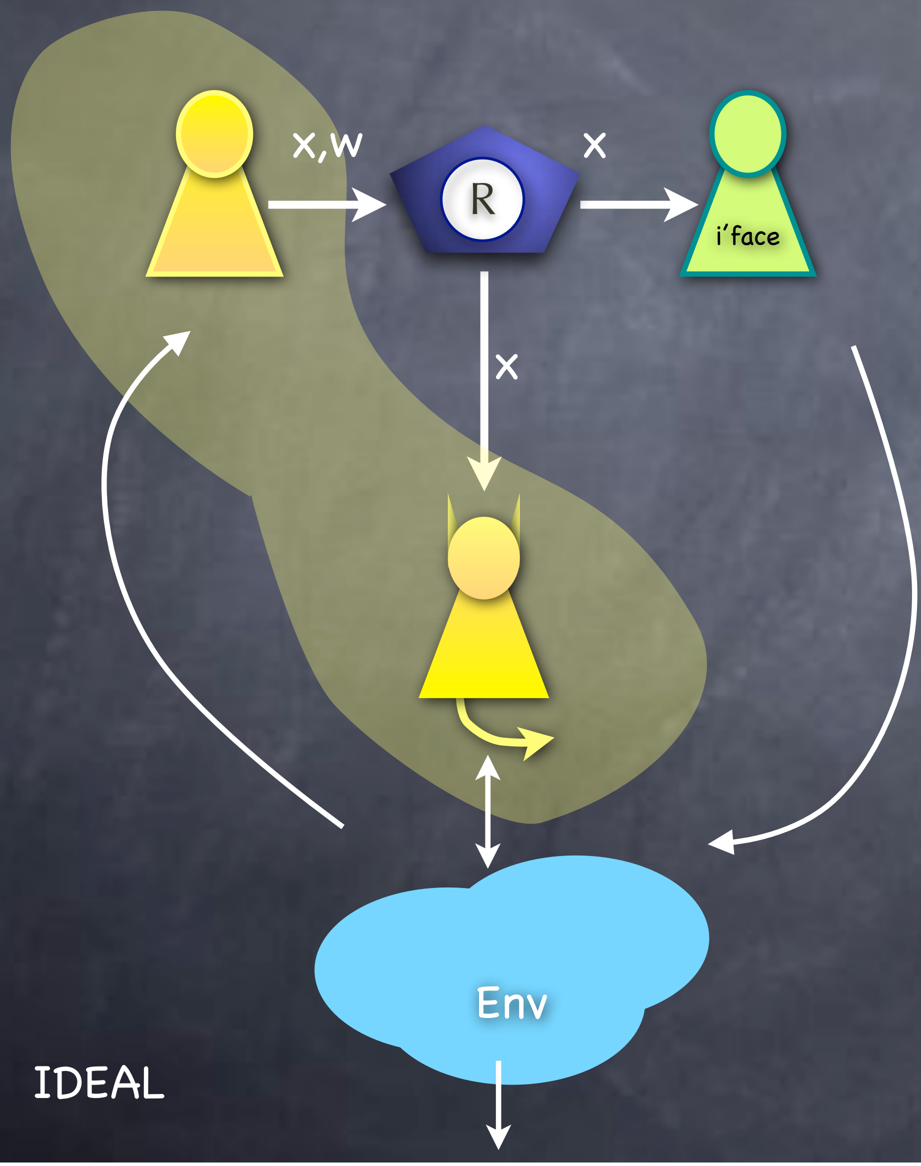
∀

output of is distributed identically in REAL and IDEAL



SIM ZK

- SIM-ZK would require simulation also when prover is corrupt



Secure (and correct) if:

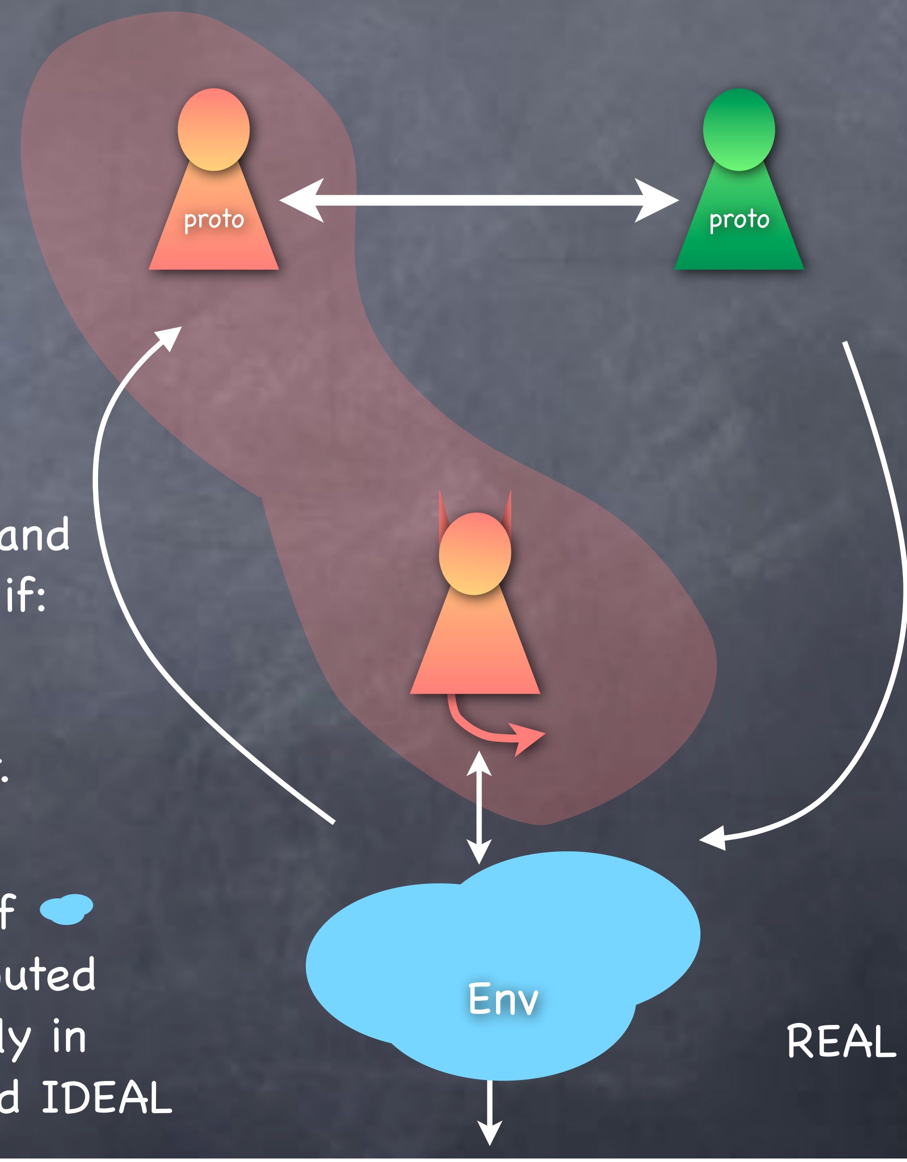
∀

∃ s.t.

∀

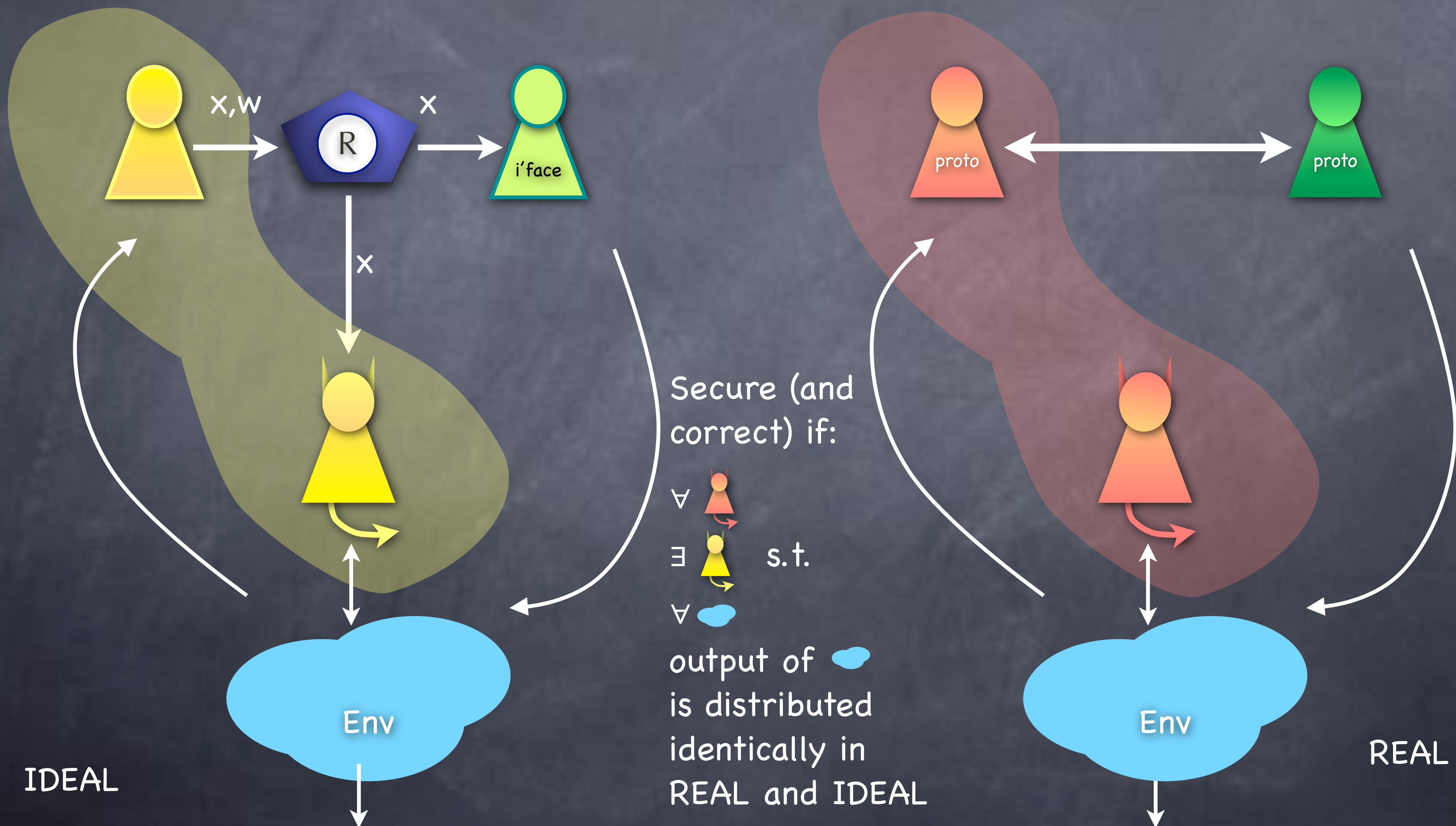
output of

is distributed identically in REAL and IDEAL



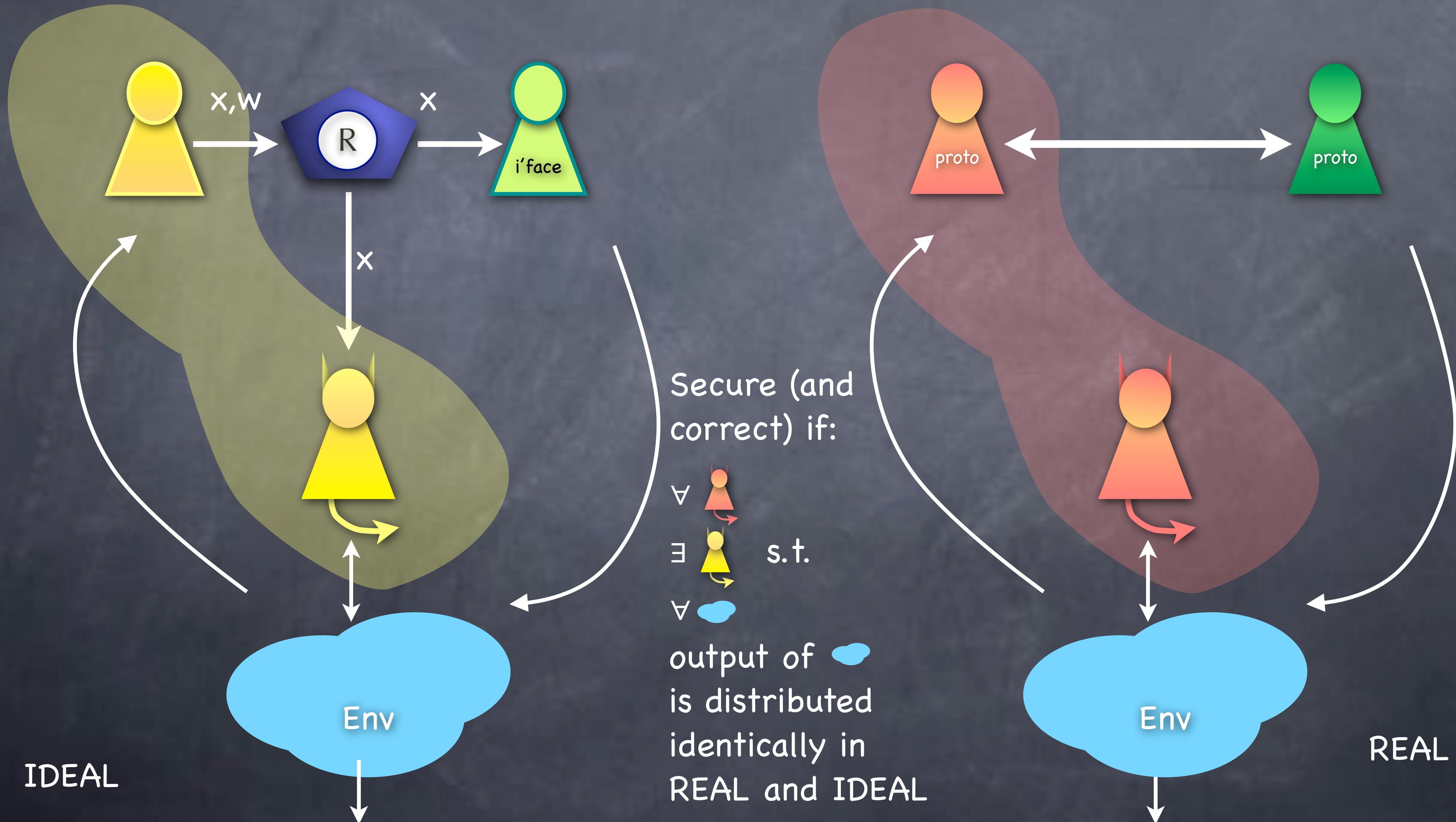
SIM ZK

- SIM-ZK would require simulation also when prover is corrupt
  - Then simulator is a witness extractor



SIM ZK

- SIM-ZK would require simulation also when prover is corrupt
  - Then simulator is a witness extractor
- Adding this (in standalone) makes it a Proof of Knowledge



Results

Results

- IP and ZK defined [GMR'85]

Results

- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]

Results

- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]
  - Assuming one-way functions exist

Results

- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]
  - Assuming one-way functions exist
- ZK for all of IP [BGGHKMR'88]

Results

- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]
  - Assuming one-way functions exist
- ZK for all of IP [BGGHKMR'88]
  - Everything that can be proven can be proven in zero-knowledge! (Assuming OWF)

Results

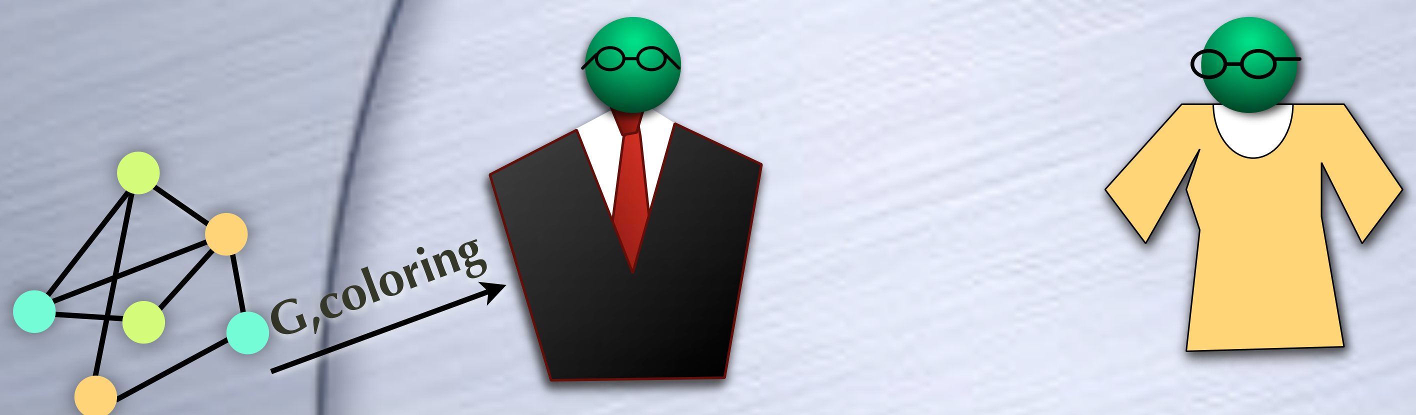
- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]
  - Assuming one-way functions exist
- ZK for all of IP [BGGHKMR'88]
  - Everything that can be proven can be proven in zero-knowledge! (Assuming OWF)
- Variants (for NP)

Results

- IP and ZK defined [GMR'85]
- ZK for all NP languages [GMW'86]
  - Assuming one-way functions exist
- ZK for all of IP [BGGHKMR'88]
  - Everything that can be proven can be proven in zero-knowledge! (Assuming OWF)
- Variants (for NP)
  - ZKPoK, Statistical ZK Arguments, O(1)-round ZK, ...

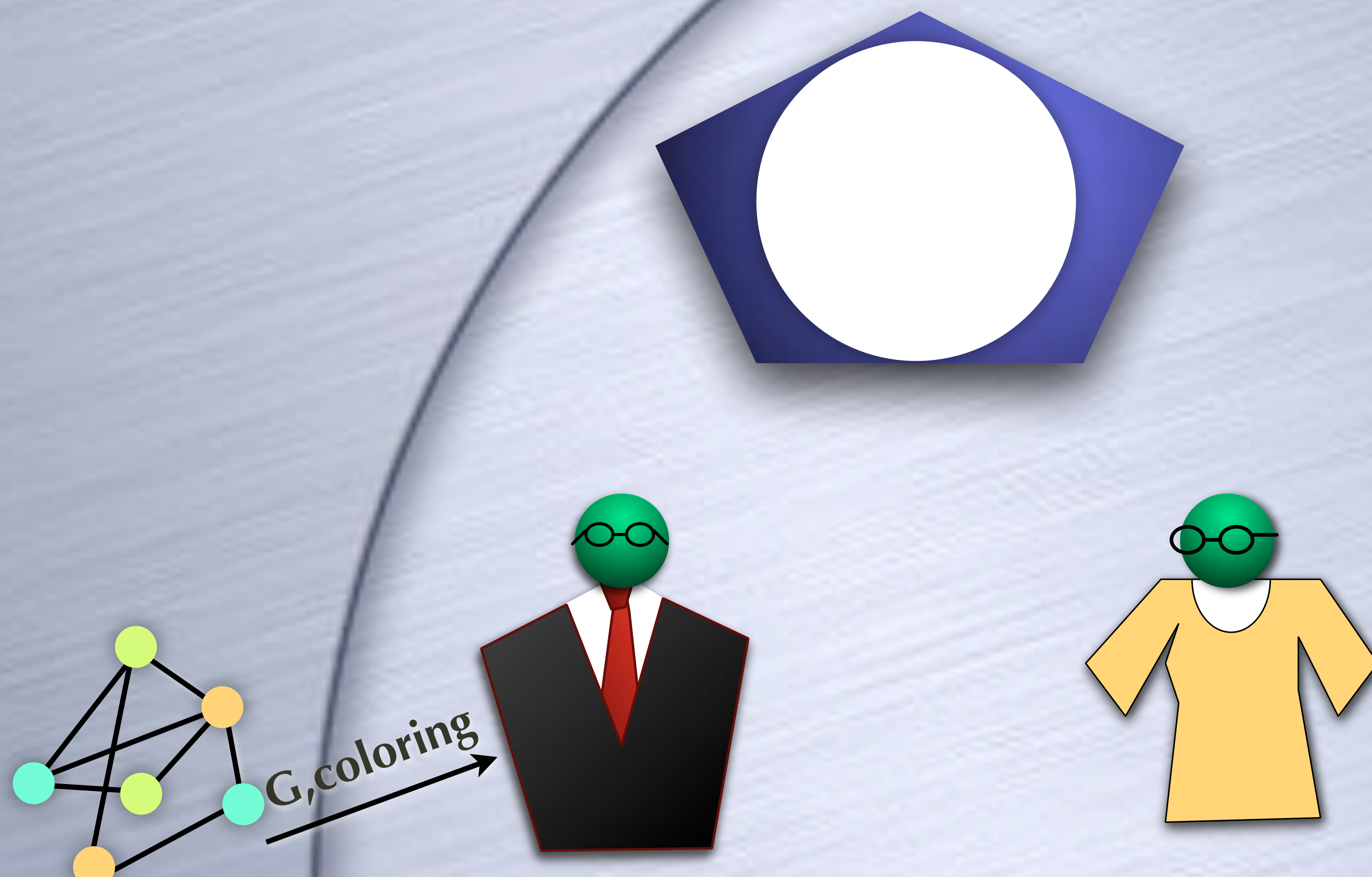
A ZK Proof for Graph Colorability

A ZK Proof for Graph Colorability



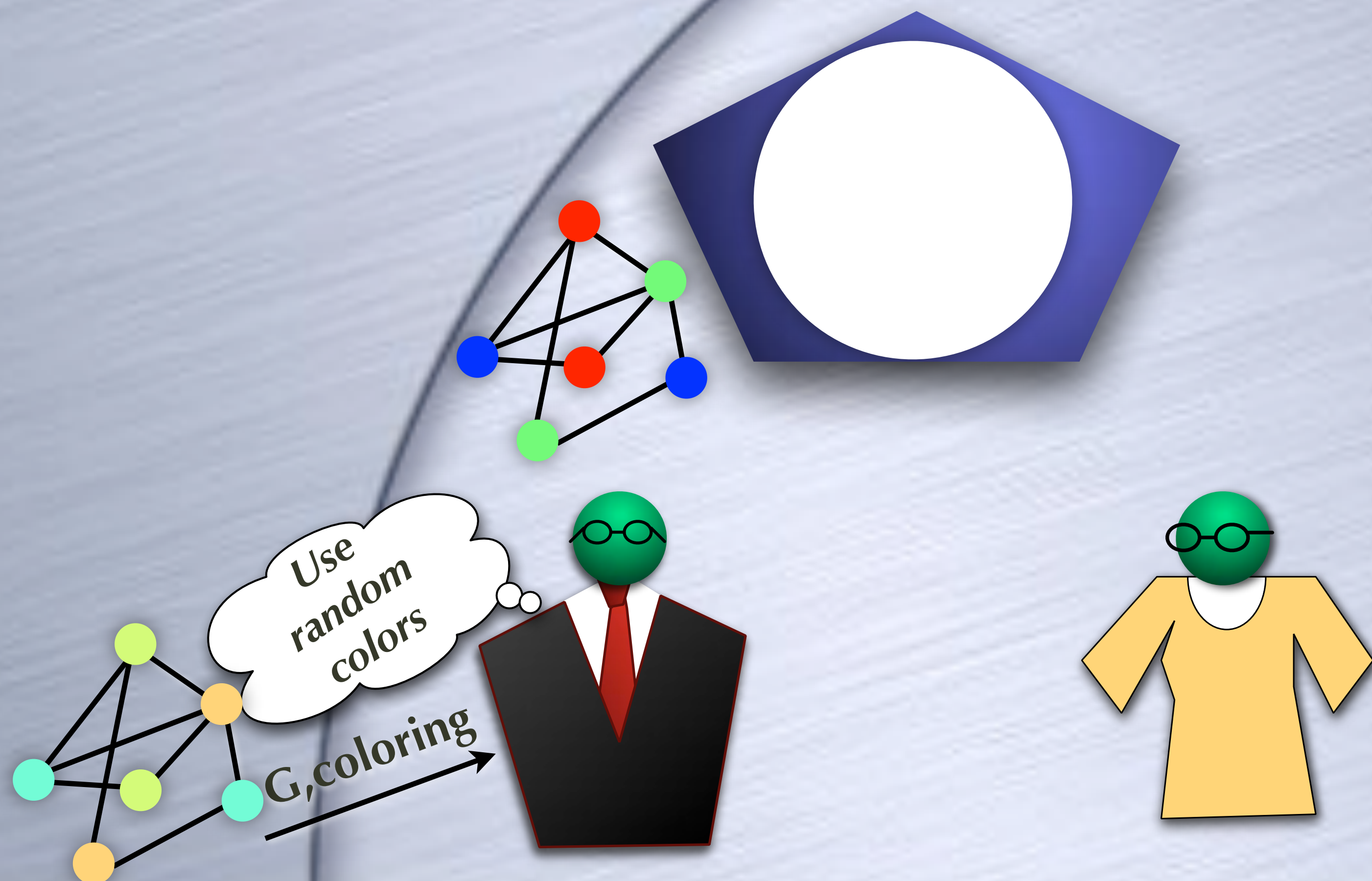
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine



A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine

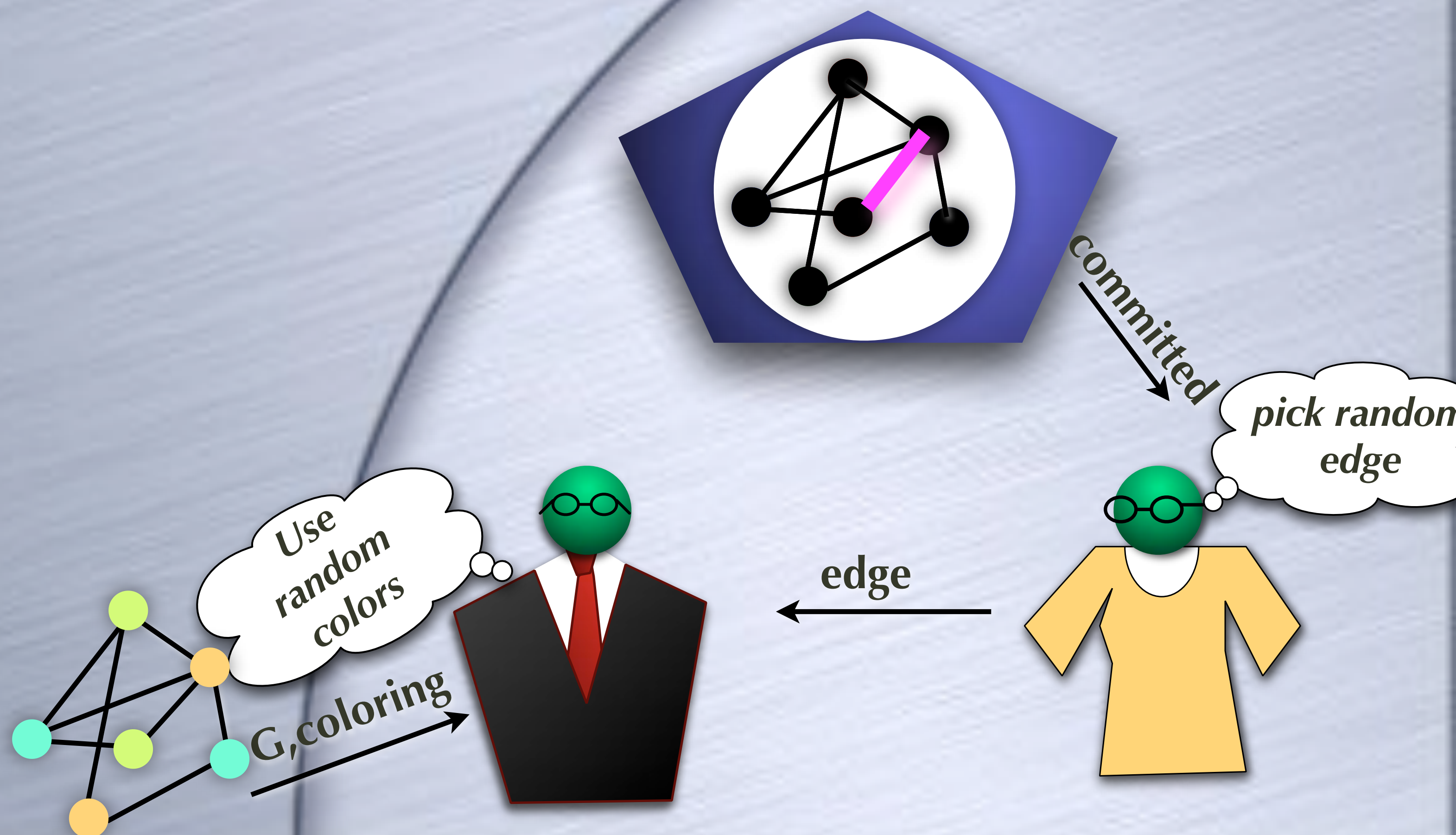


A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine

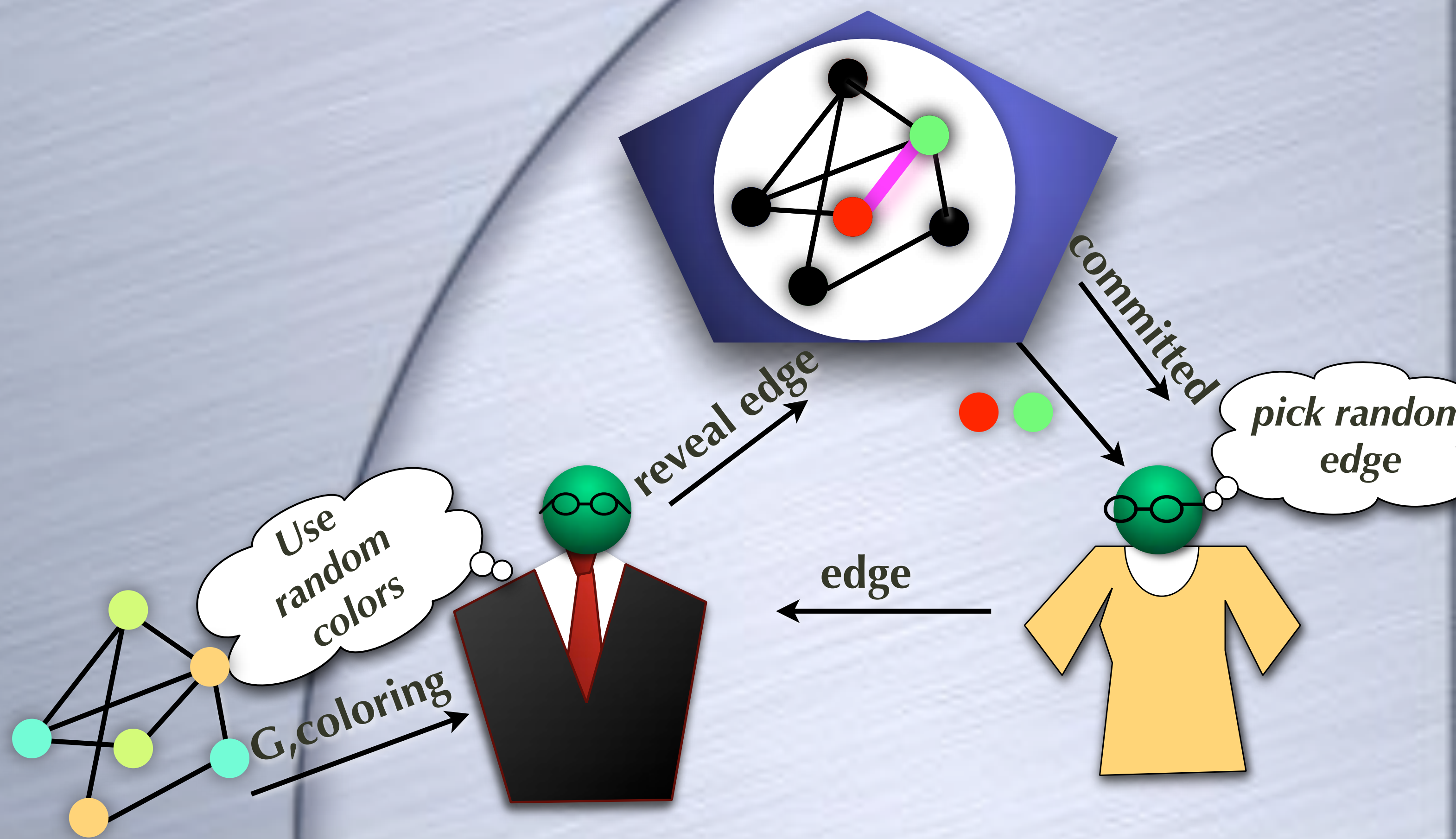
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine



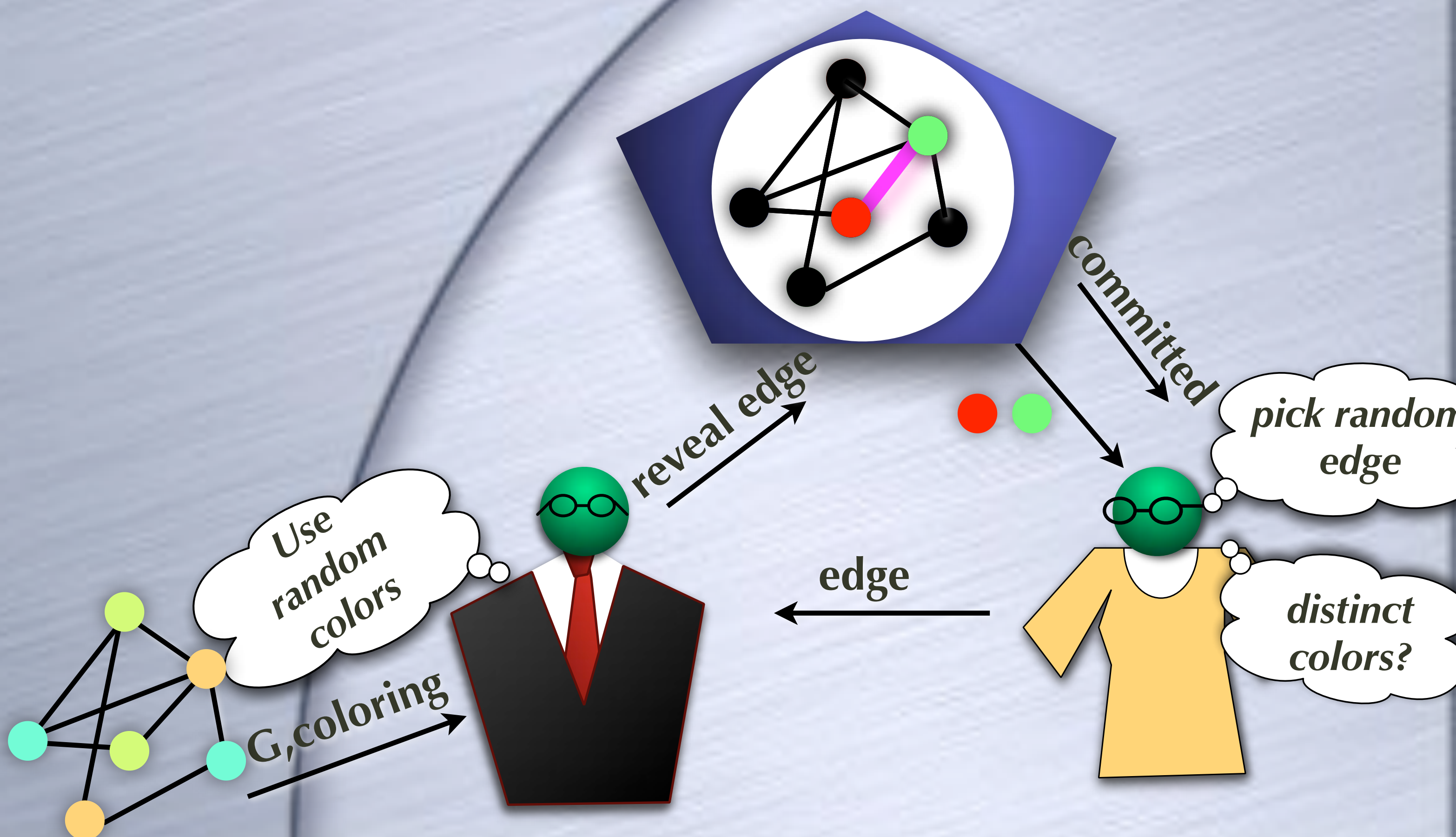
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine



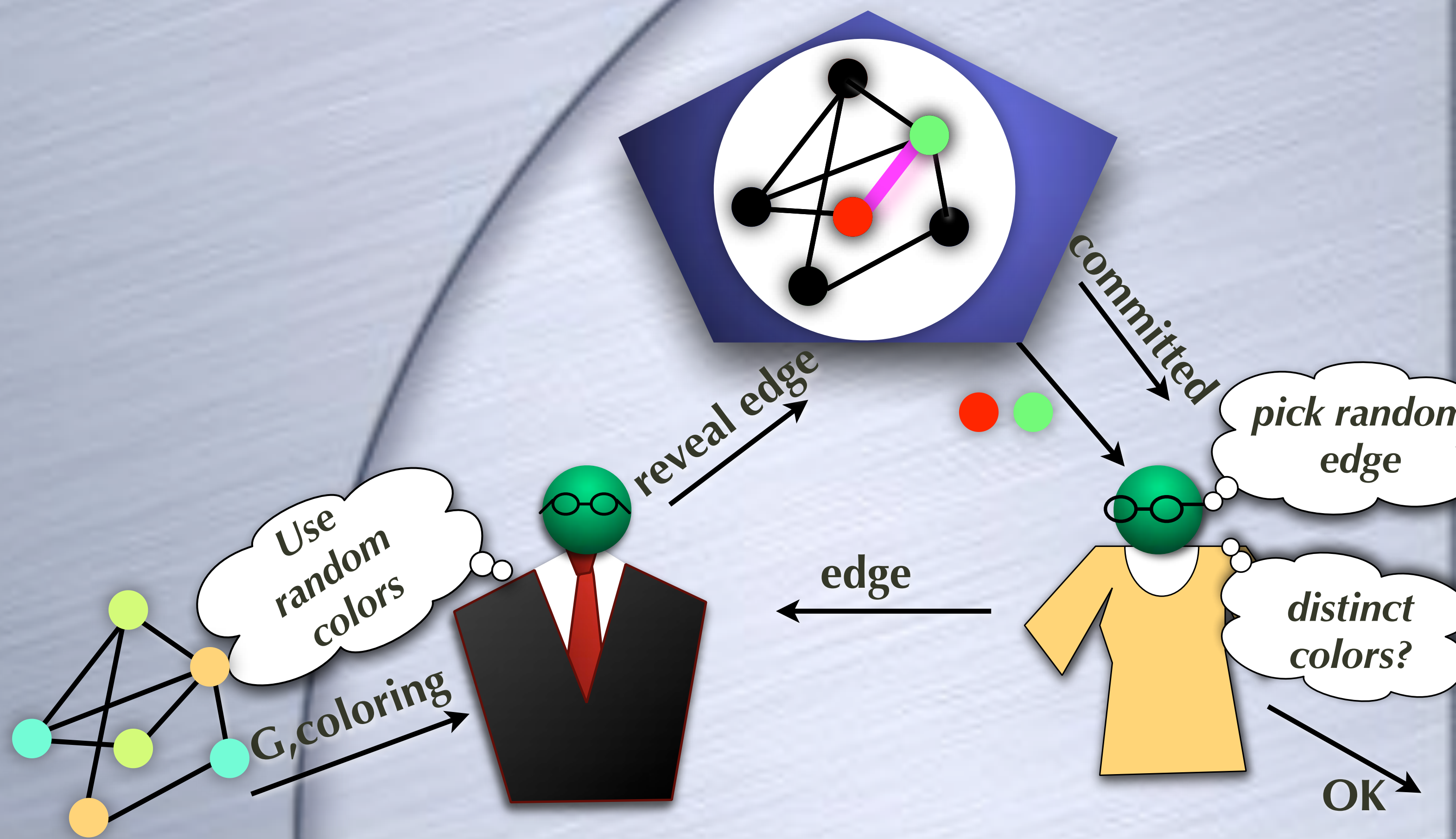
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine



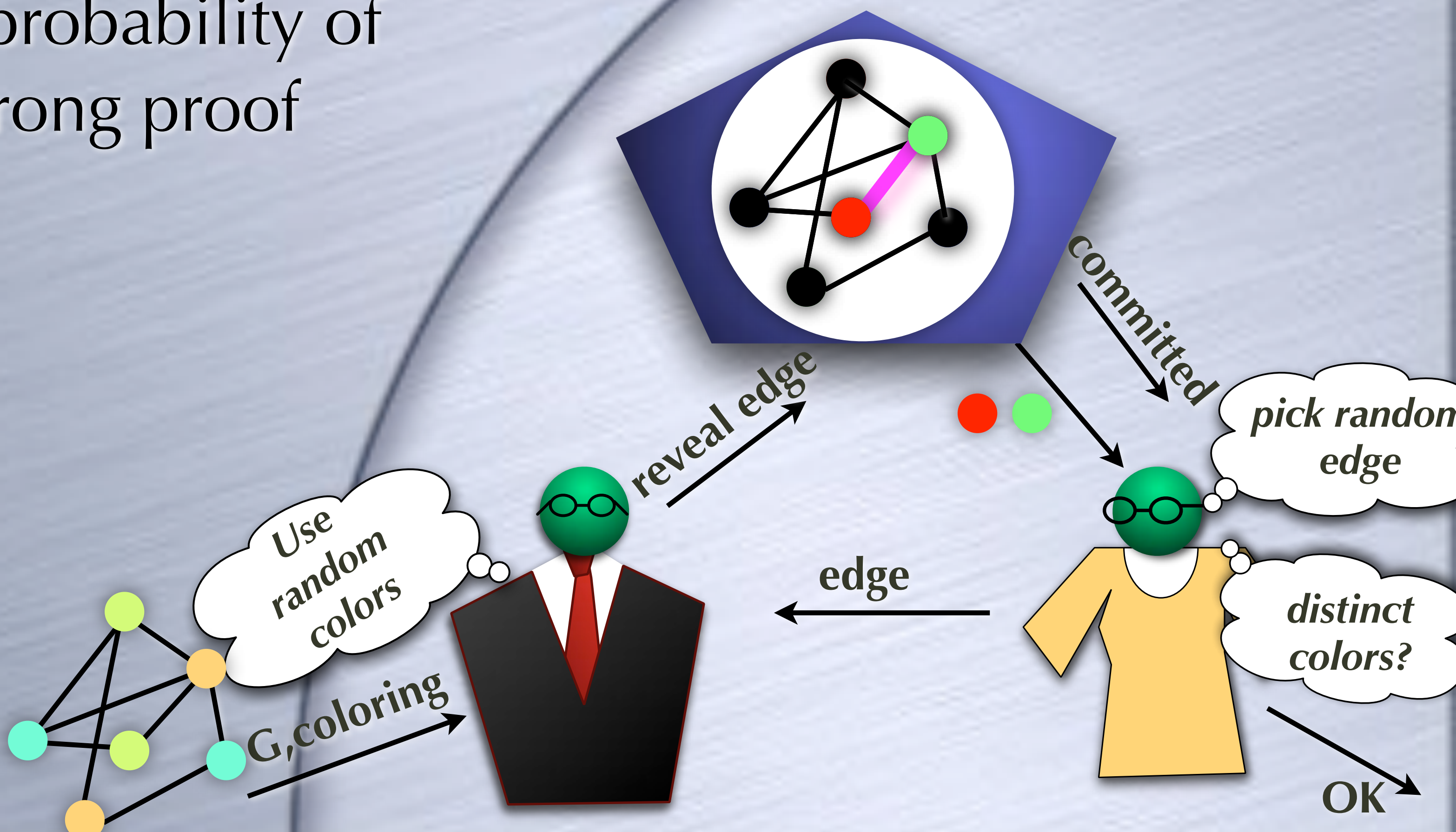
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine



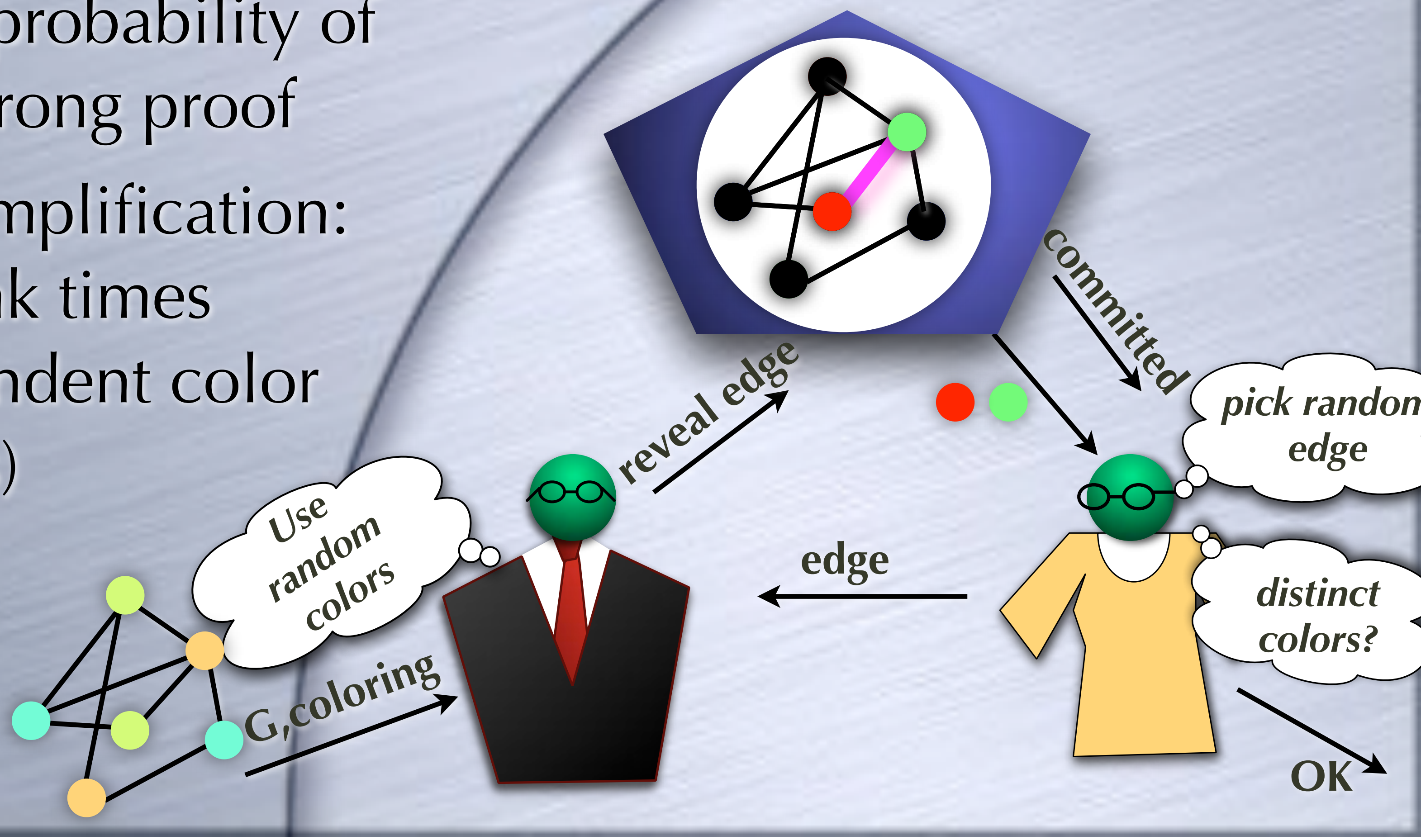
A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine
- At least 1/m probability of catching a wrong proof



A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine
- At least 1/m probability of catching a wrong proof
- Soundness amplification: Repeat say mk times (with independent color permutations)



A Commitment Protocol

A Commitment Protocol

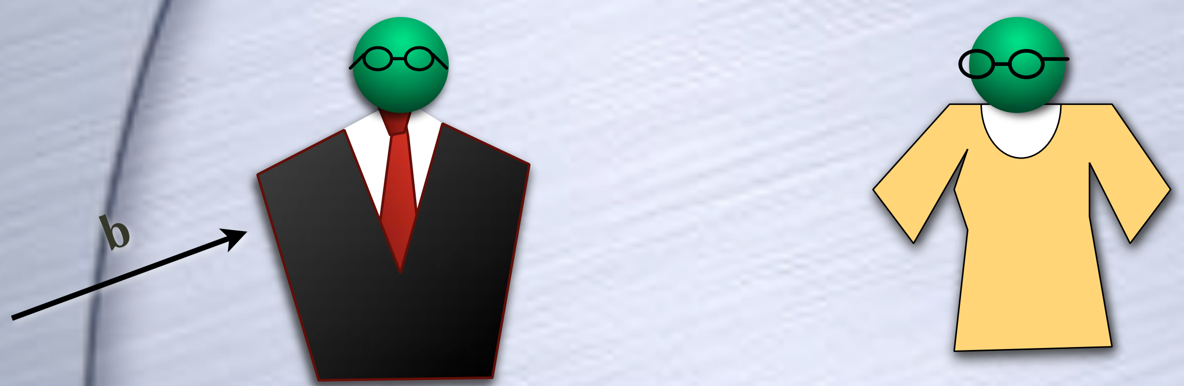
- Uses a OWP f and a hardcore predicate for it B

A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding

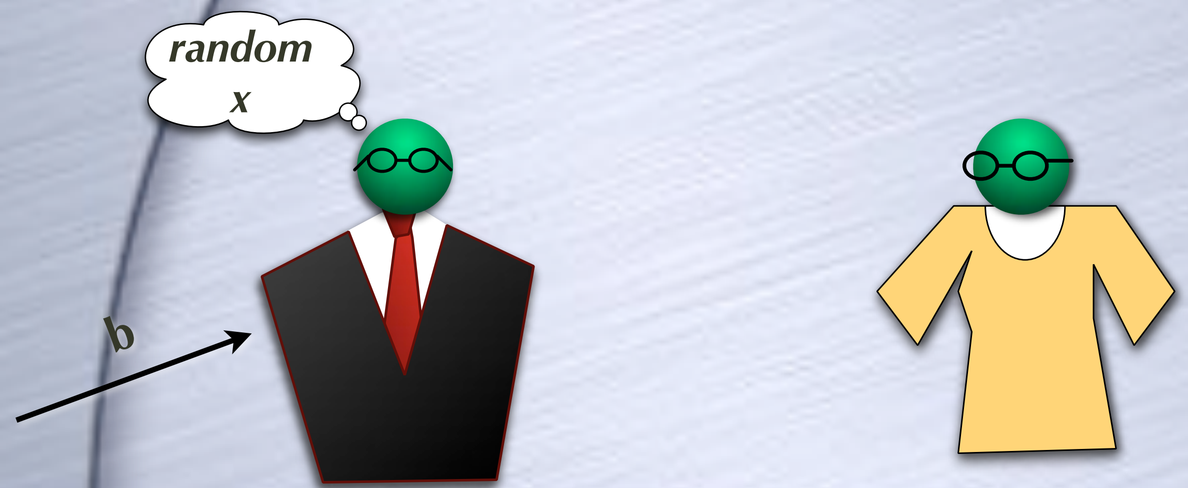
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



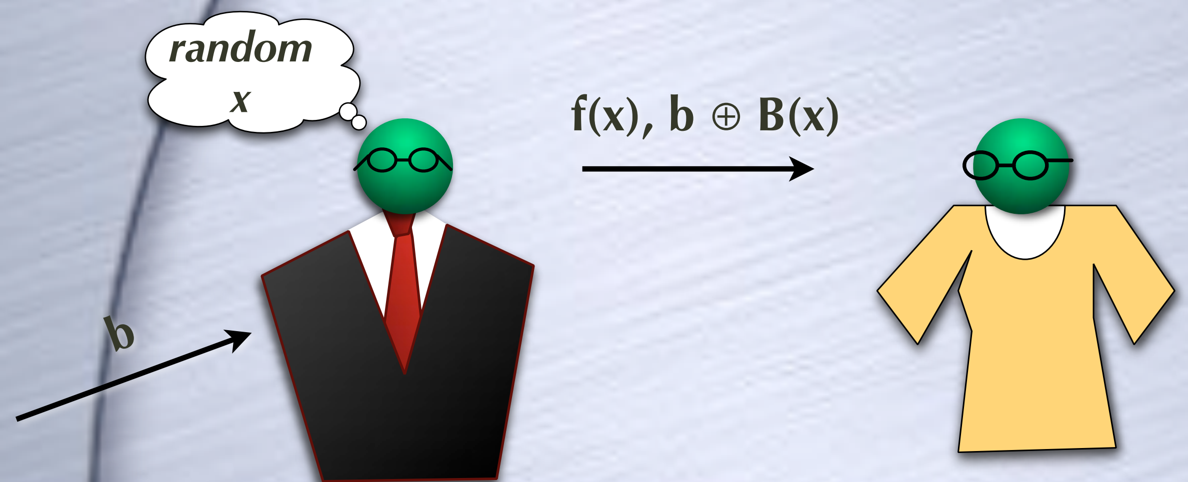
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



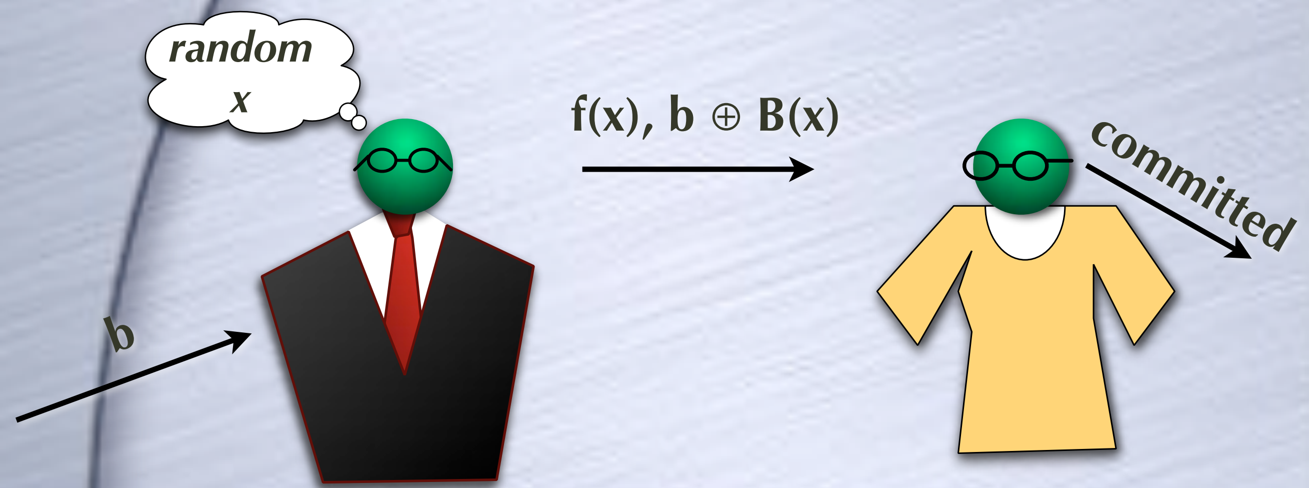
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



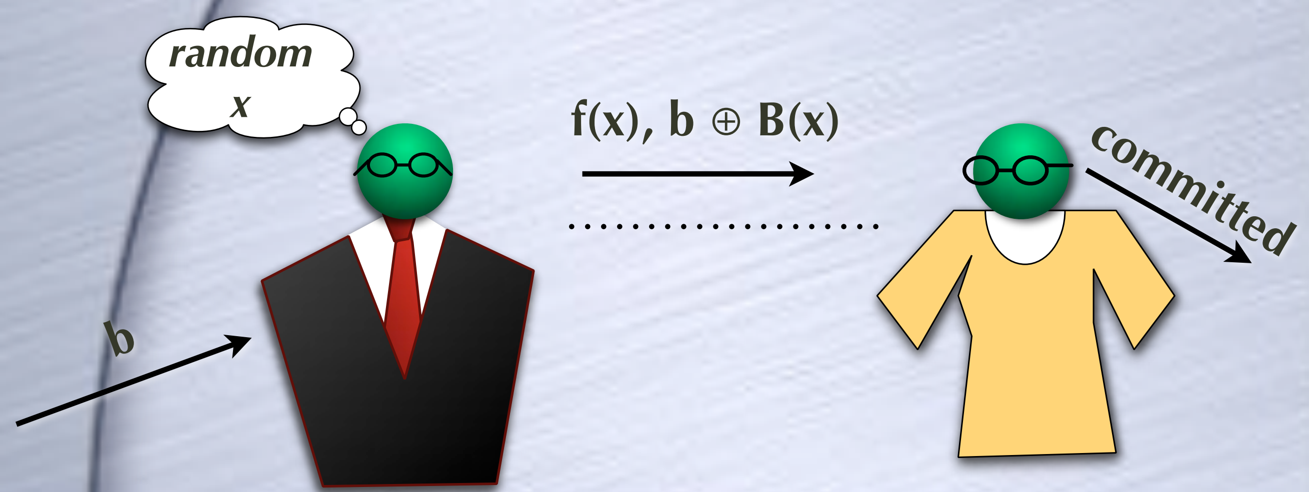
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



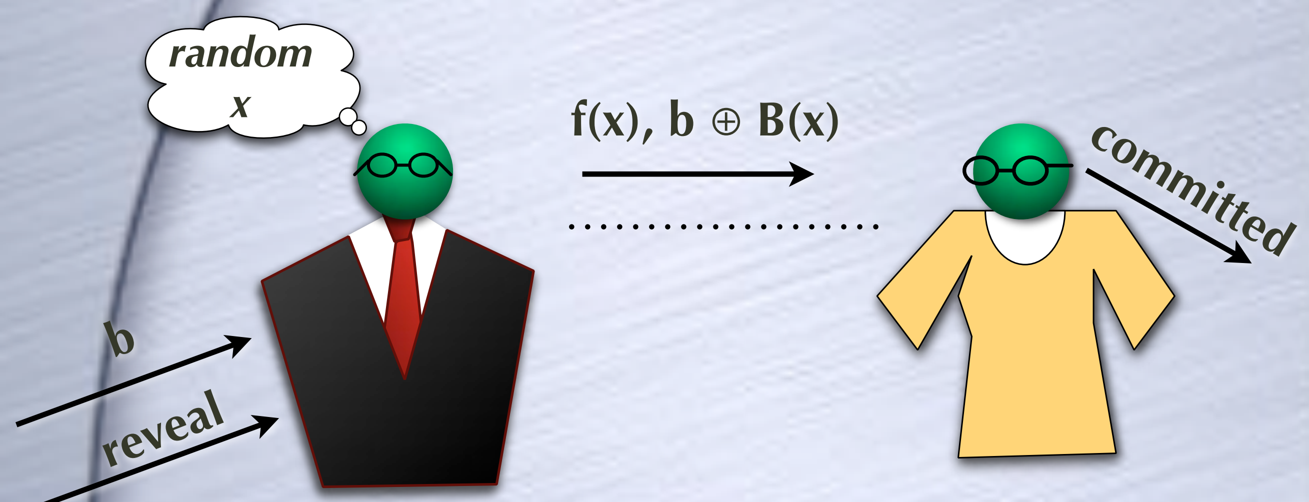
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



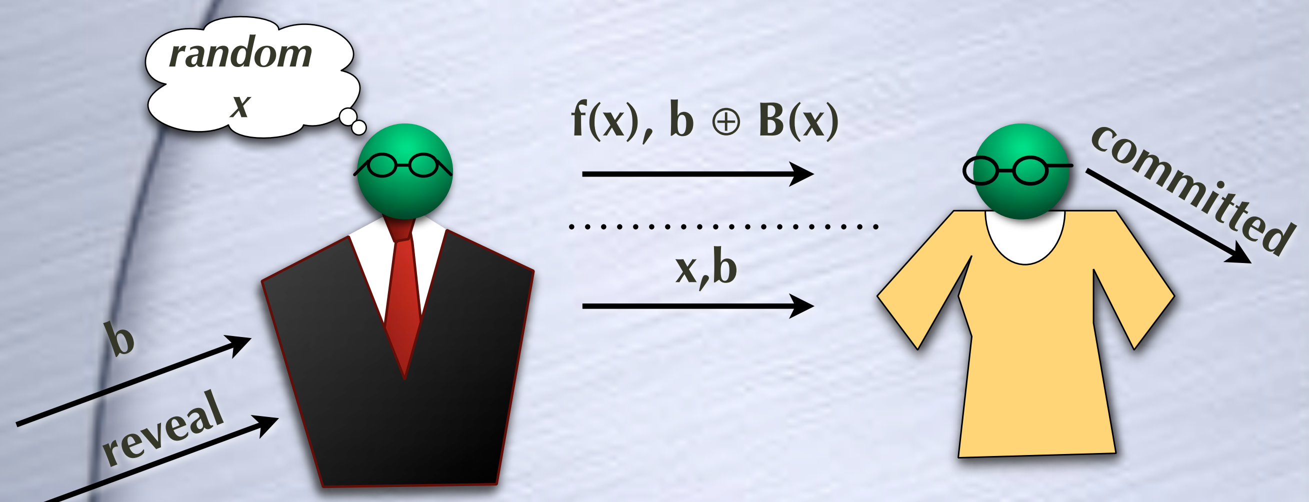
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



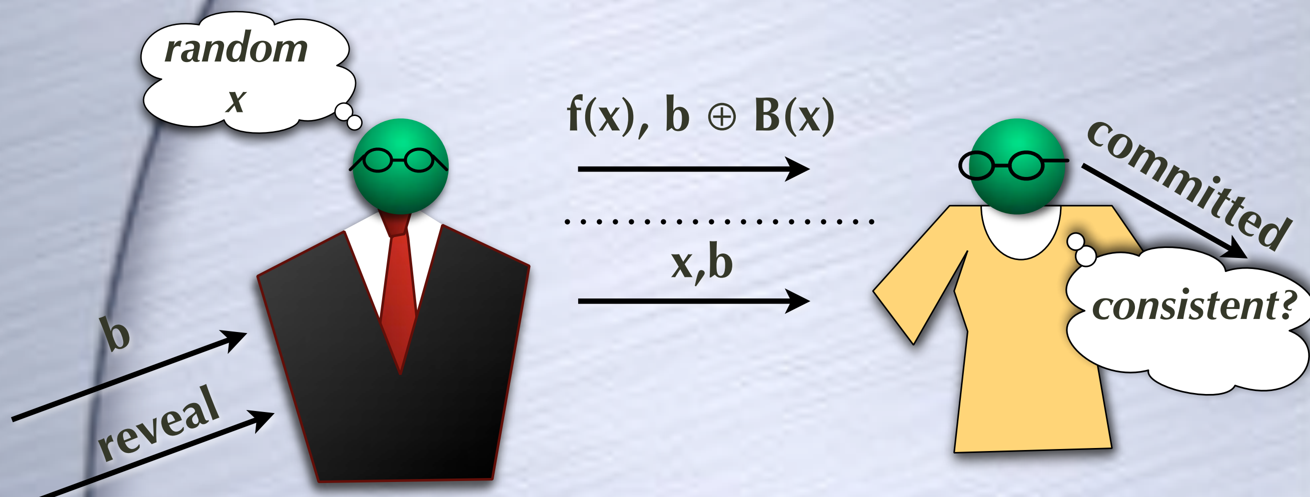
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



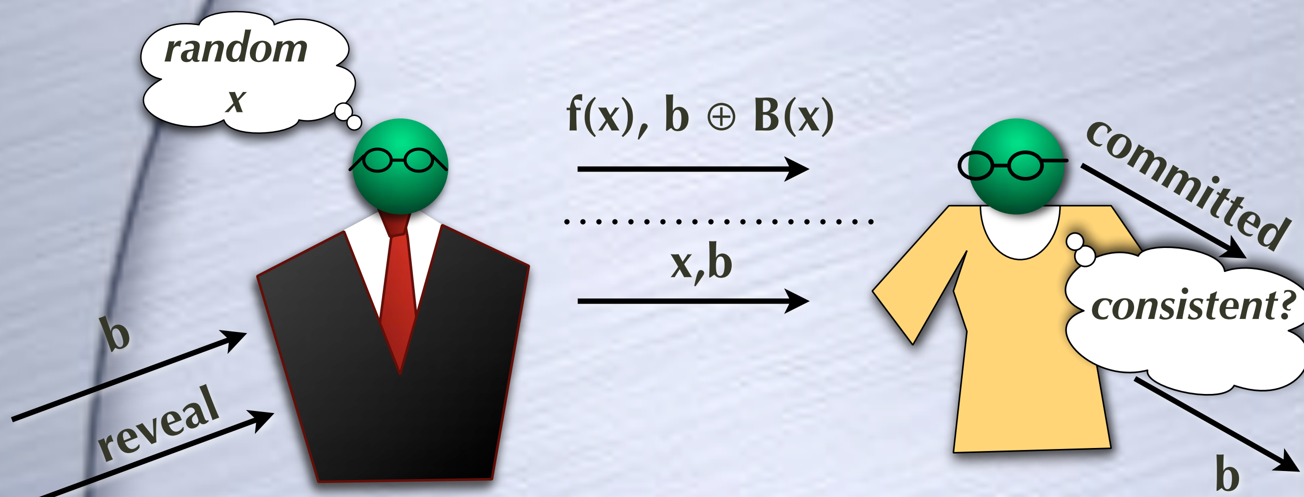
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



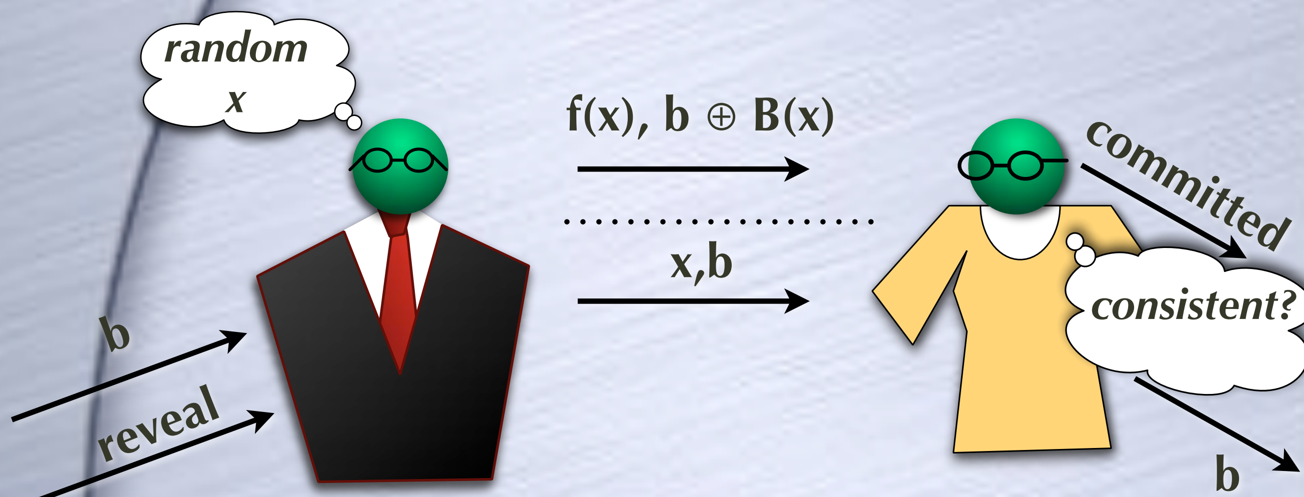
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding



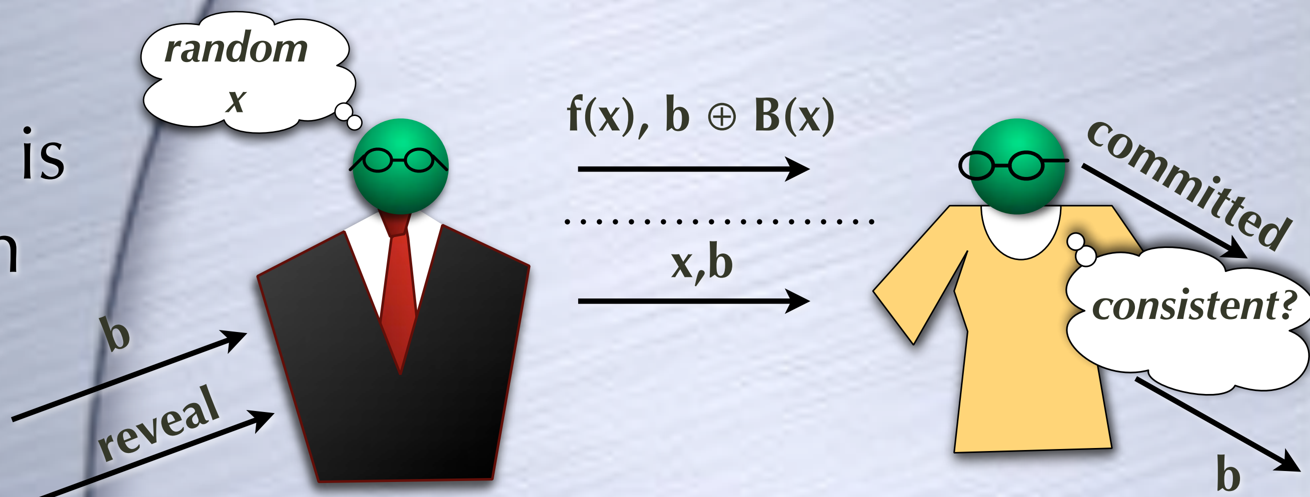
A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding
- Perfectly binding because f is a permutation



A Commitment Protocol

- Uses a OWP f and a hardcore predicate for it B
- Satisfies only classical (IND) security, in terms of hiding and binding
- Perfectly binding because f is a permutation
- Hiding because B(x) is pseudorandom given f(x)



ZK Proofs: What for?

ZK Proofs: What for?

- Authentication

ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge

ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols

ZK Proofs: What for?

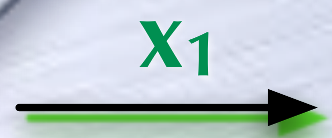
- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce “honest behavior” in protocols

ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed

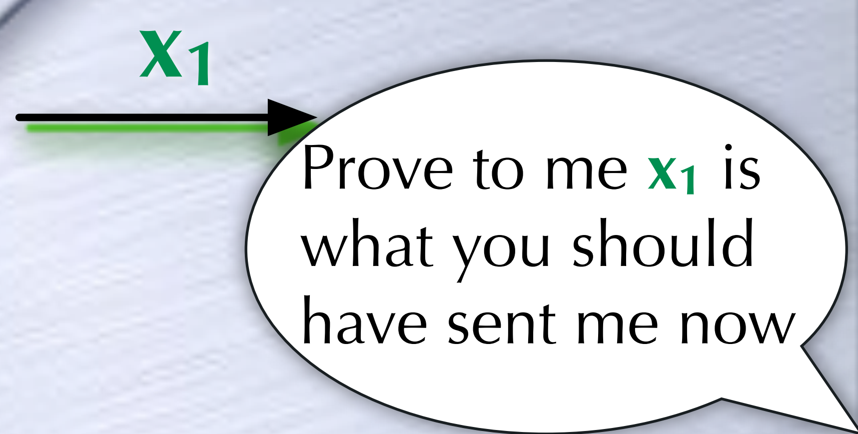
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed



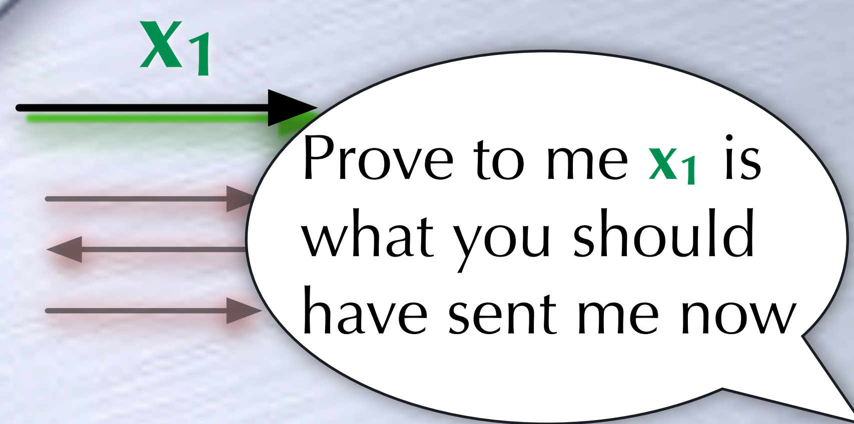
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed



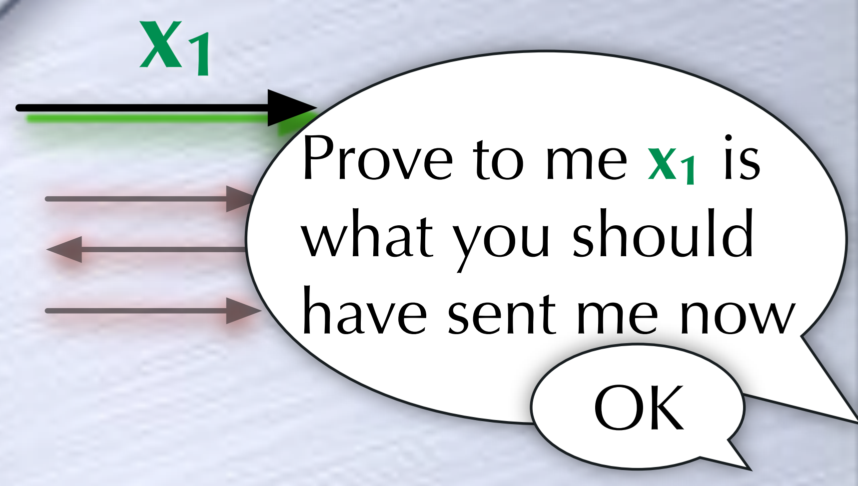
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce “honest behavior” in protocols
  - At each step prove in ZK it was done as prescribed



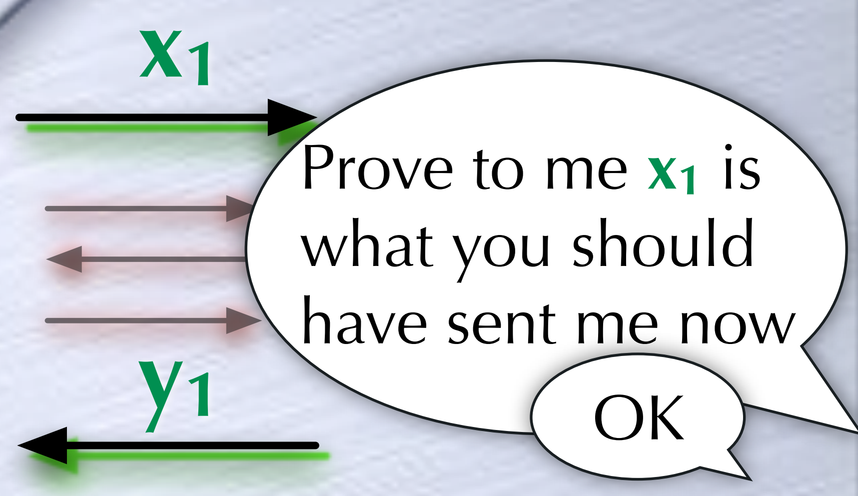
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce “honest behavior” in protocols
  - At each step prove in ZK it was done as prescribed



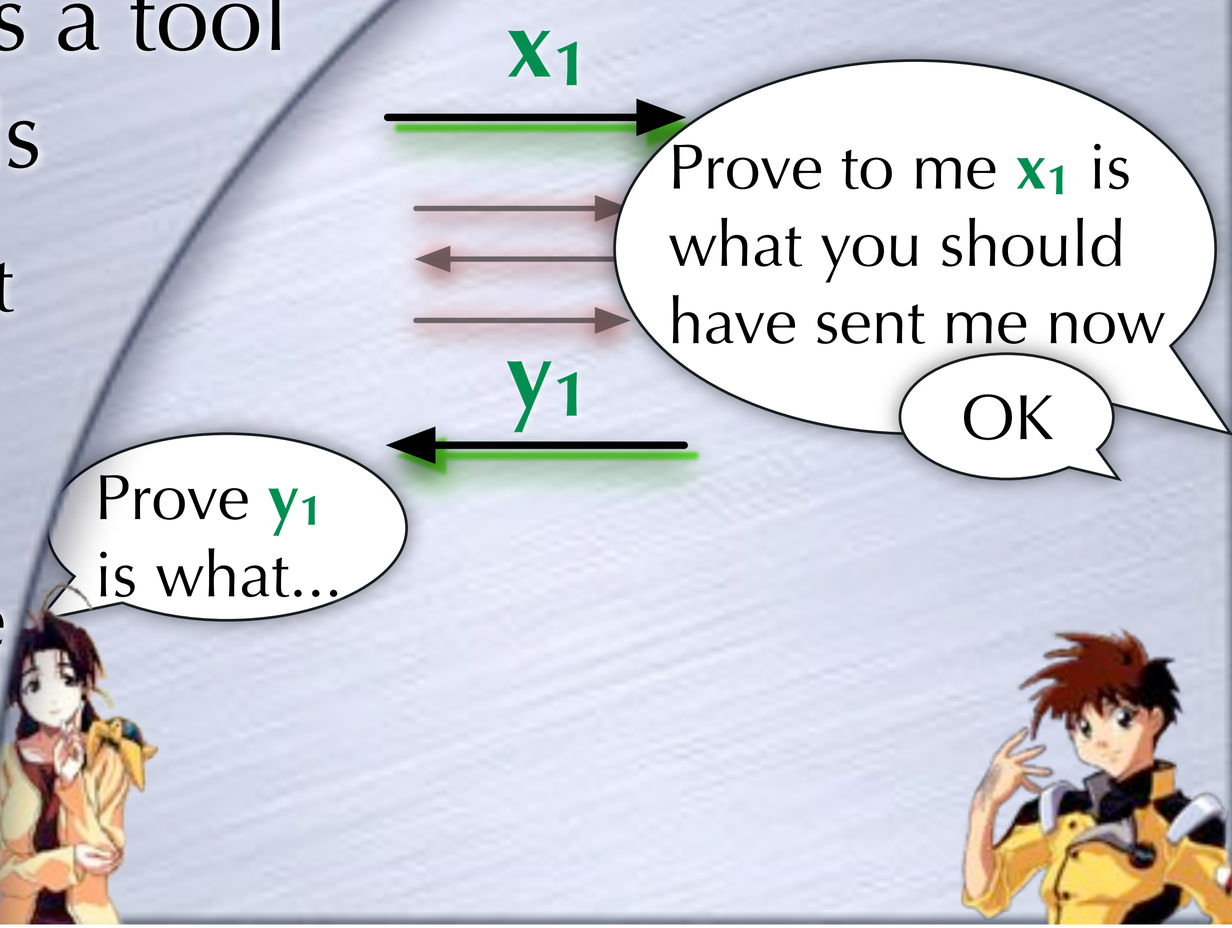
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed



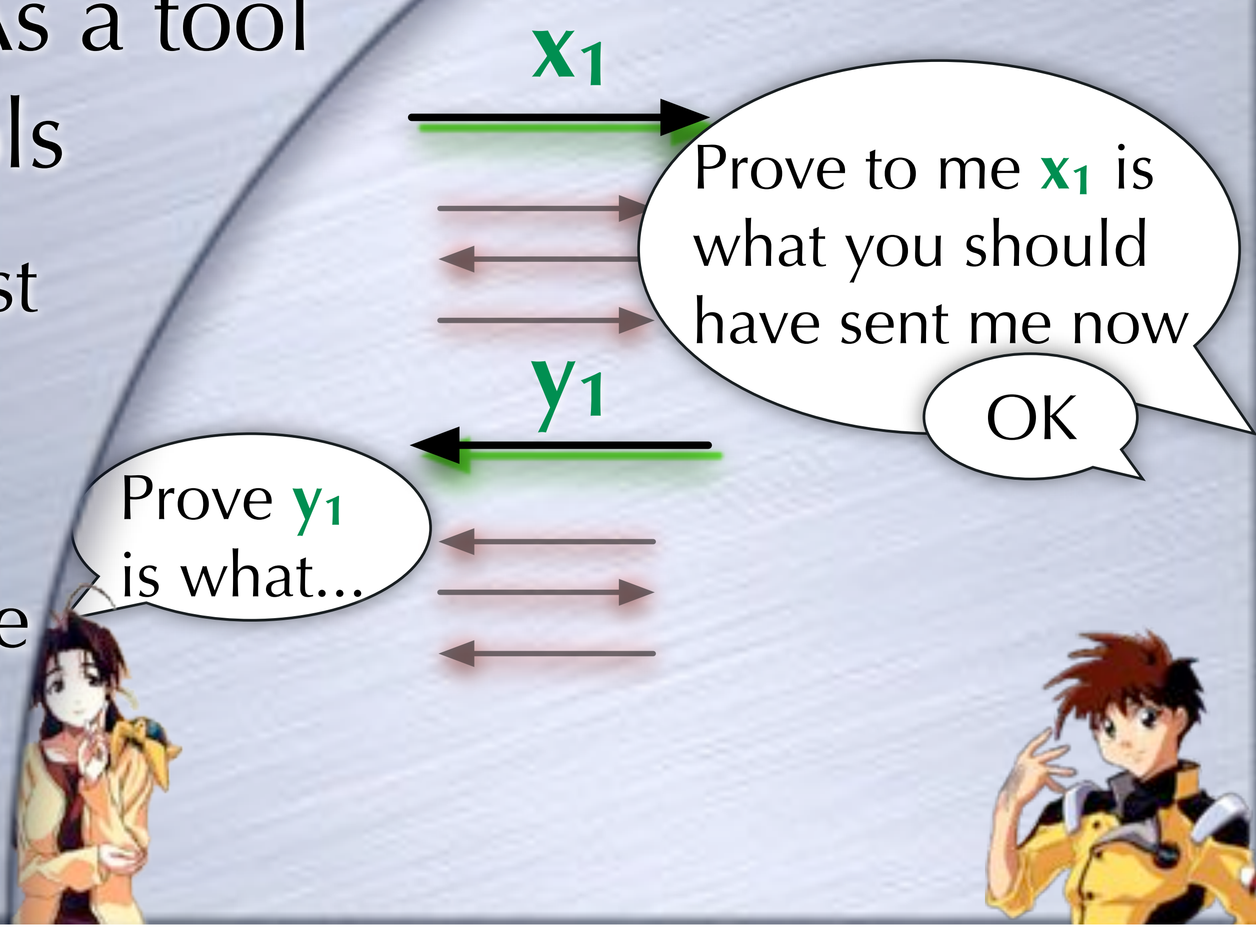
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed



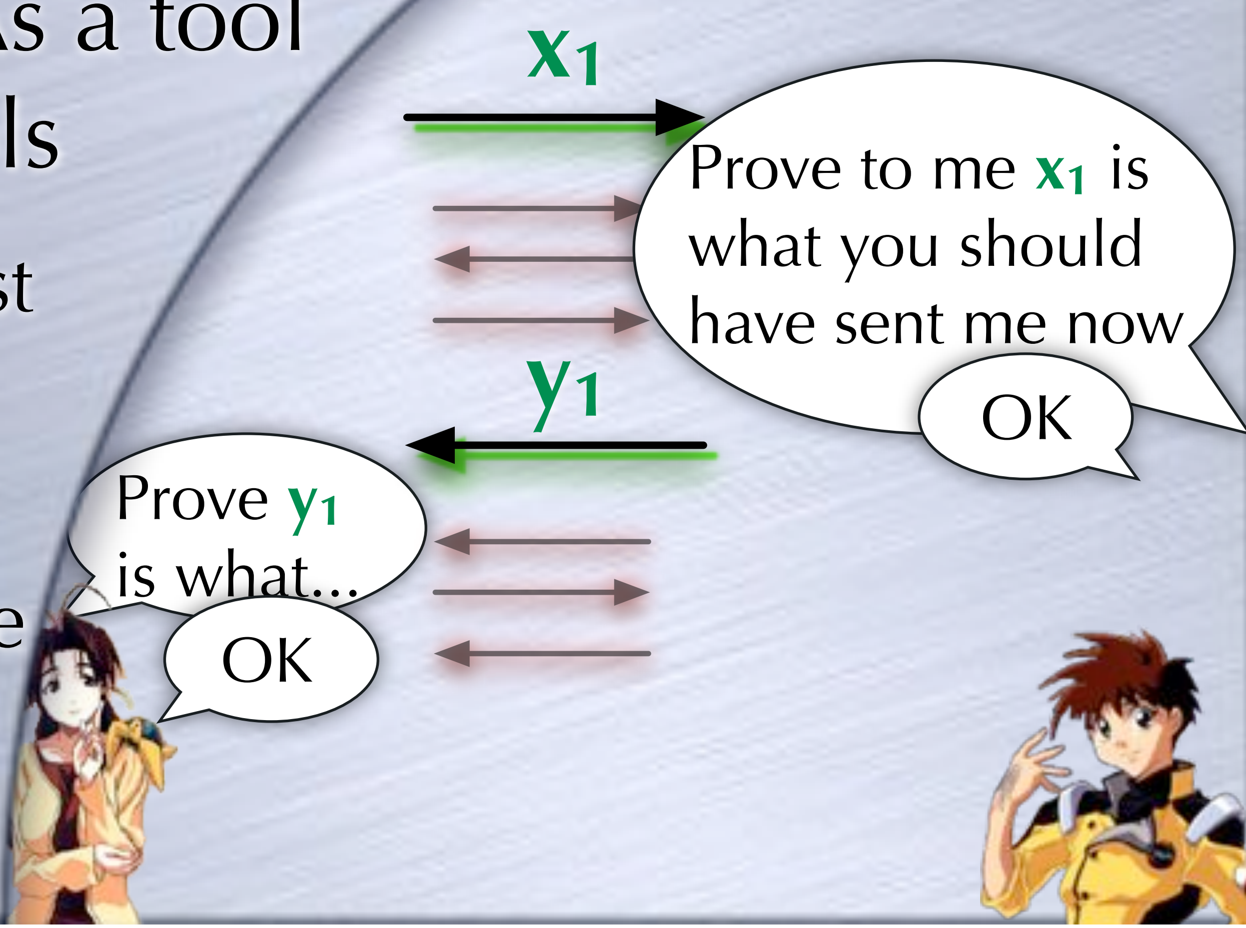
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce “honest behavior” in protocols
  - At each step prove in ZK it was done as prescribed



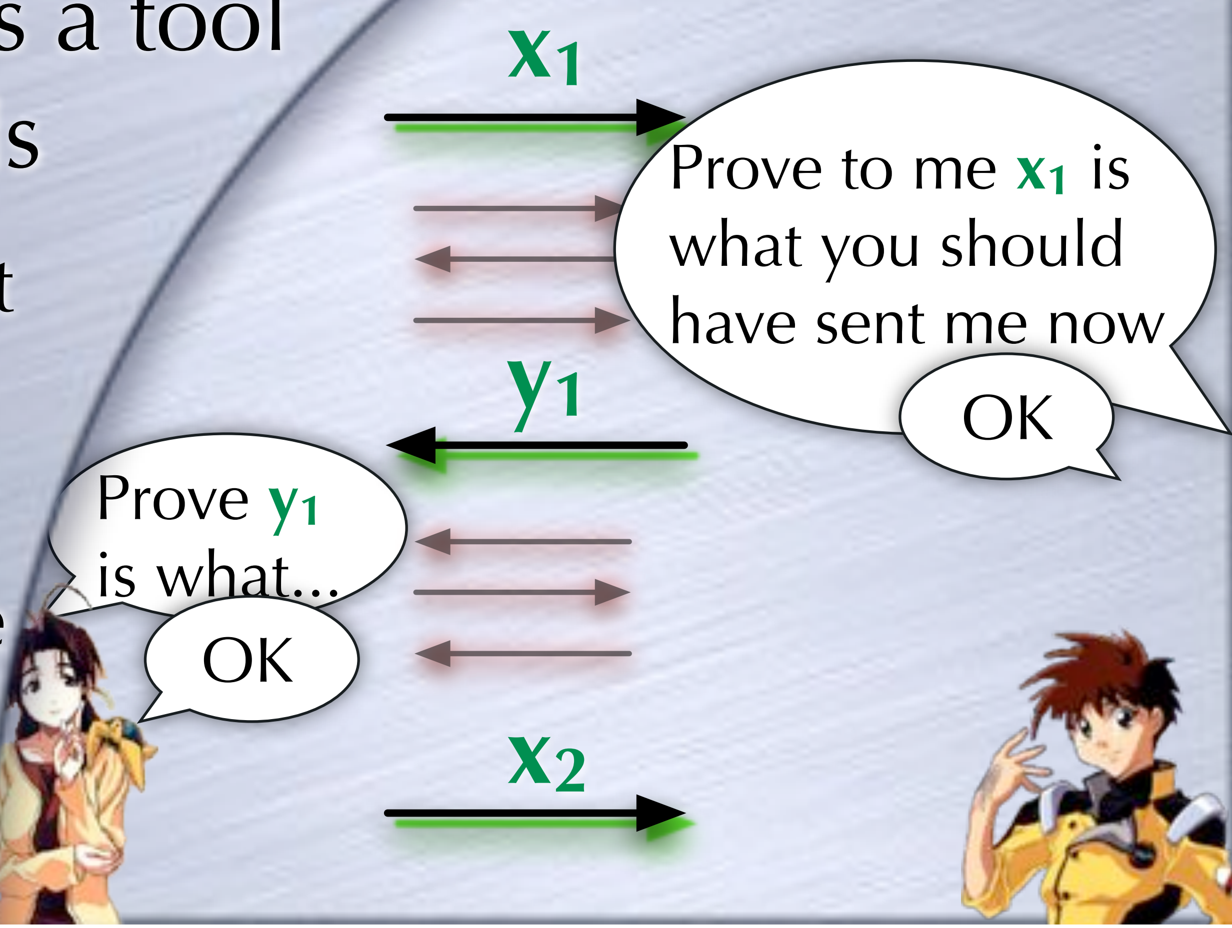
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed



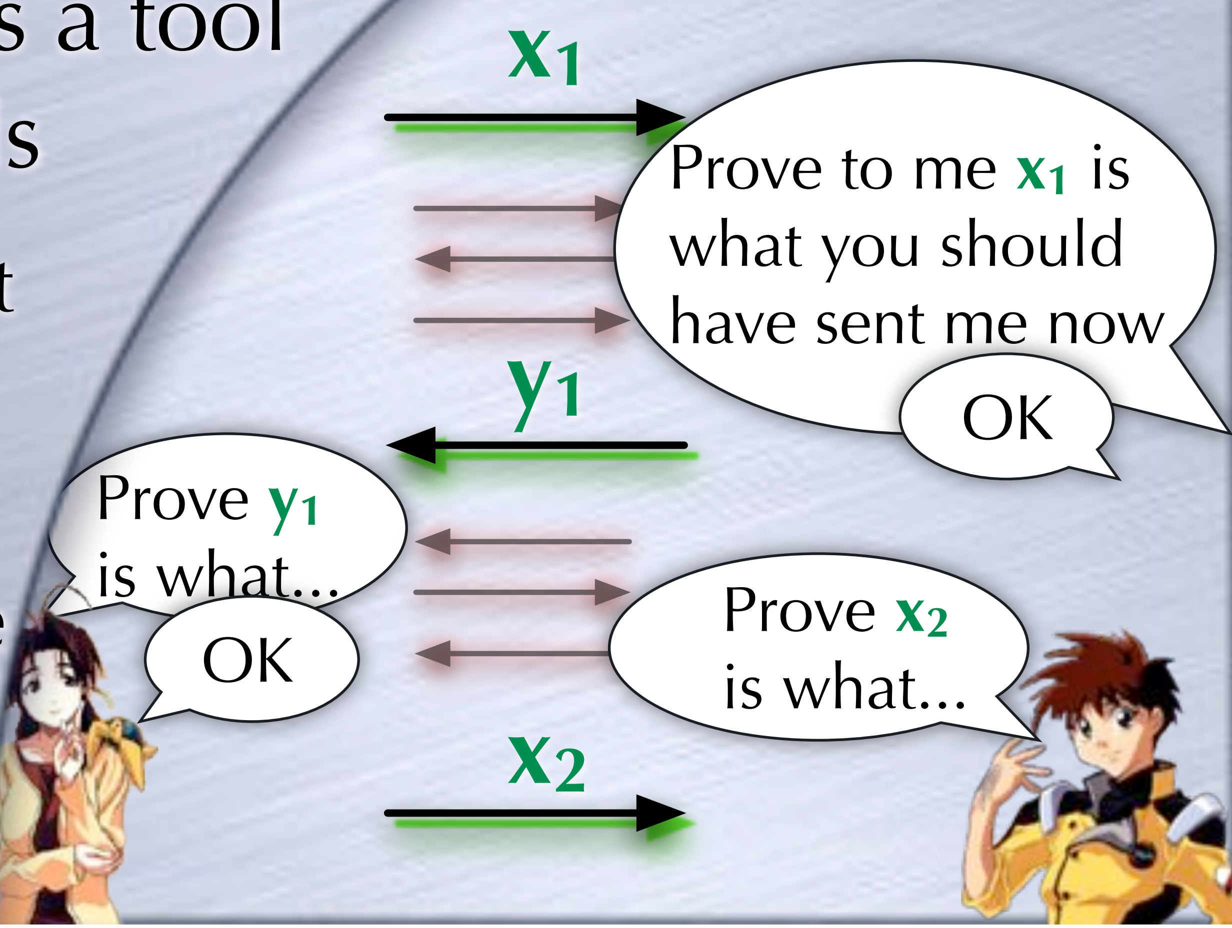
ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed

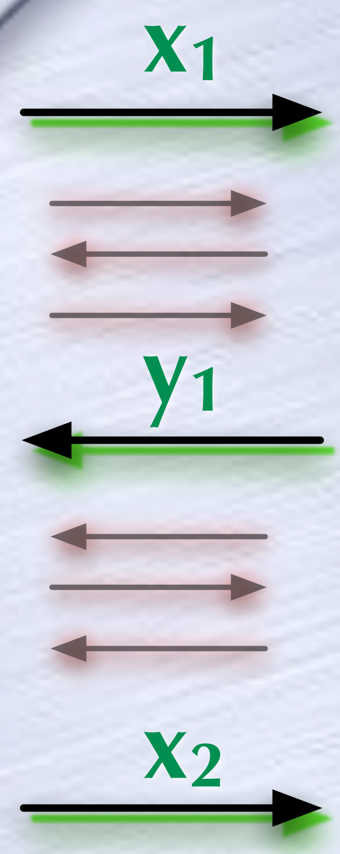


ZK Proofs: What for?

- Authentication
  - Using ZK Proof of Knowledge
- Canonical use: As a tool in larger protocols
  - To enforce "honest behavior" in protocols
  - At each step prove in ZK it was done as prescribed

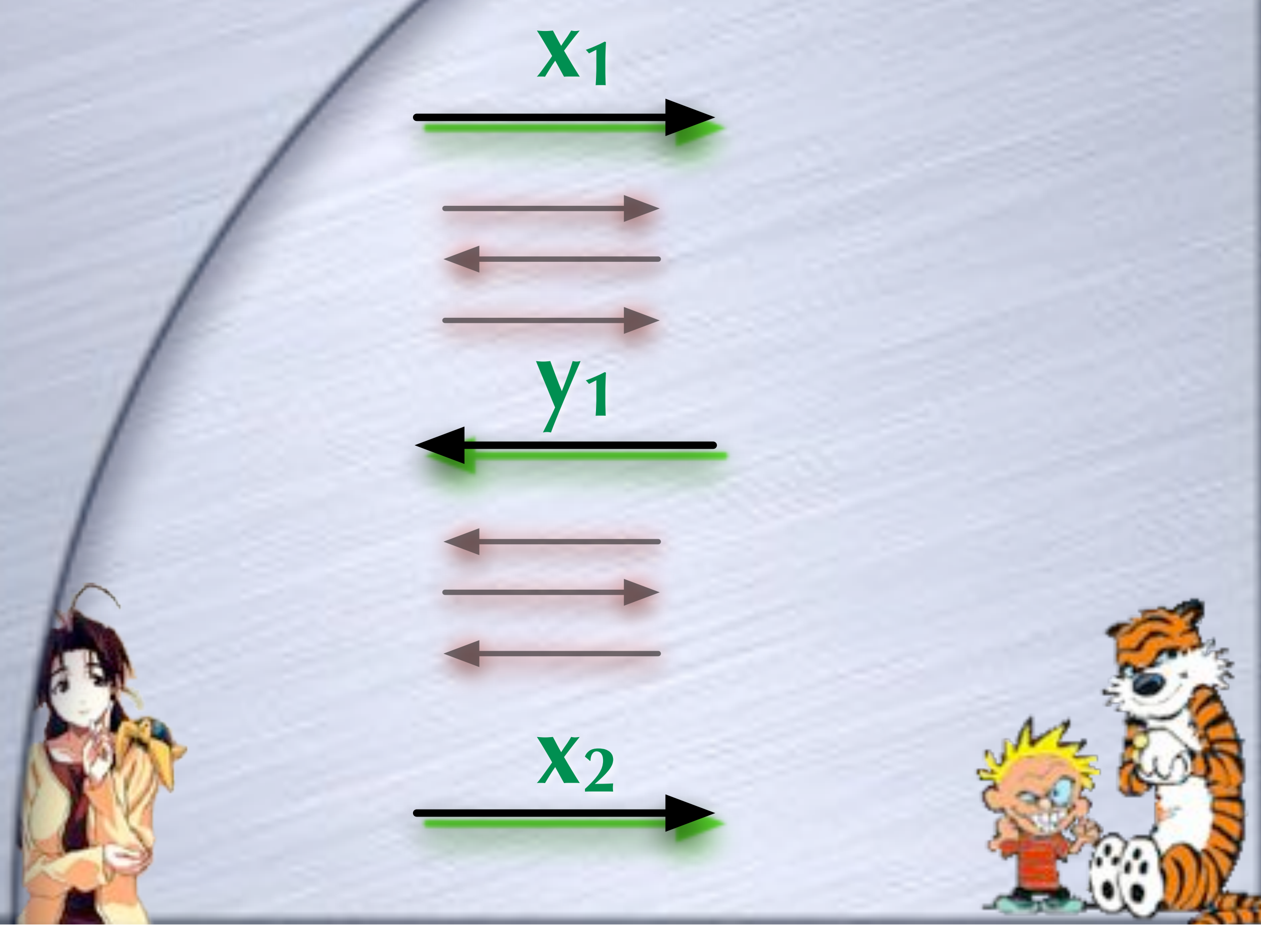


Does it fit in?



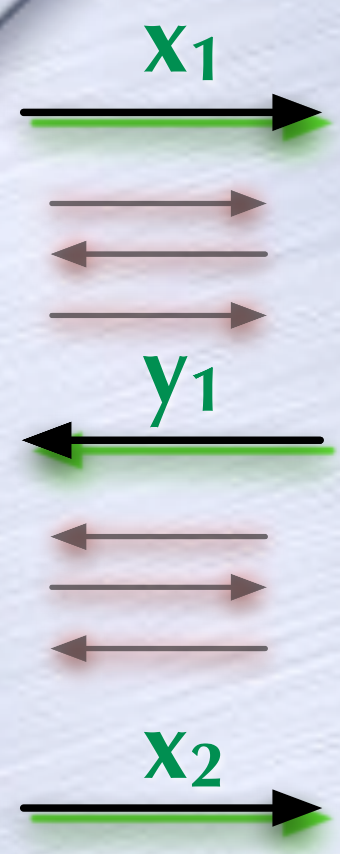
Does it fit in?

- Does the proof stay ZK in the big picture?



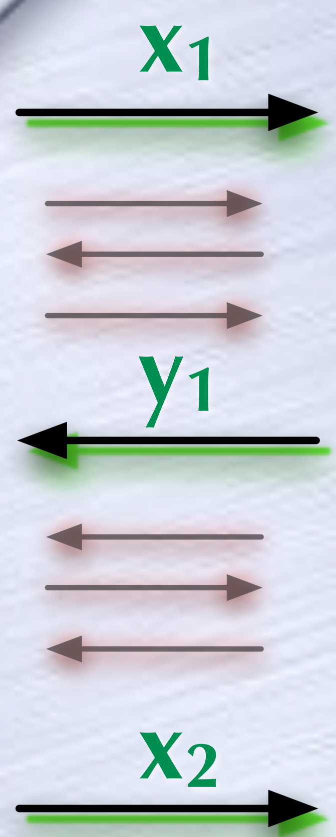
Does it fit in?

- Does the proof stay ZK in the big picture?
  - Composition



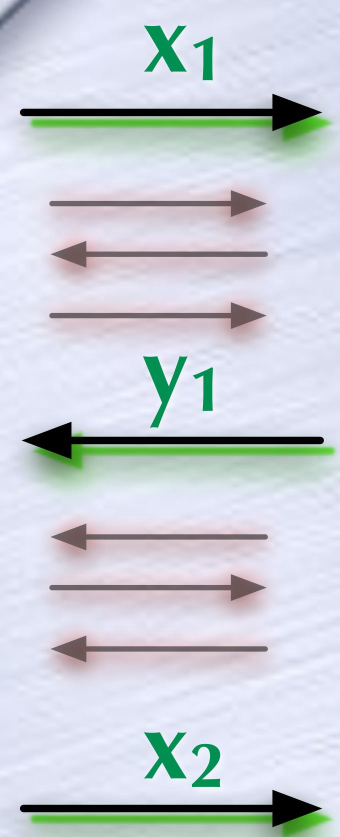
Does it fit in?

- Does the proof stay ZK in the big picture?
  - Composition
    - Several issues: auxiliary information from previous runs, concurrency issues, malleability/man-in-the-middle



Does it fit in?

- Does the proof stay ZK in the big picture?
  - Composition
    - Several issues: auxiliary information from previous runs, concurrency issues, malleability/man-in-the-middle
  - In general, to allow composition more complicated protocols



An IND-security Notion

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x,w_0,w_1) and prover uses (x,w_b) for a random b. Adversary has negligible advantage in guessing b.

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x,w_0,w_1) and prover uses (x,w_b) for a random b. Adversary has negligible advantage in guessing b.
    - A ZK proof is always WI, but not vice-versa

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x,w_0,w_1) and prover uses (x,w_b) for a random b. Adversary has negligible advantage in guessing b.
    - A ZK proof is always WI, but not vice-versa
- WI Proofs used as components inside larger protocols

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x, w_0, w_1) and prover uses (x, w_b) for a random b. Adversary has negligible advantage in guessing b.
    - A ZK proof is always WI, but not vice-versa
- WI Proofs used as components inside larger protocols
  - Sometimes with certain other useful properties

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x,w_0,w_1) and prover uses (x,w_b) for a random b. Adversary has negligible advantage in guessing b.
    - A ZK proof is always WI, but not vice-versa
- WI Proofs used as components inside larger protocols
  - Sometimes with certain other useful properties
    - e.g. WI-PoK, "Sigma protocols"

An IND-security Notion

- ZK (as opposed to SIM-ZK/ZK-PoK) weakens soundness guarantee
- A weakening of ZK property: Witness Indistinguishability (WI)
  - Adversarial verifier gives (x,w_0,w_1) and prover uses (x,w_b) for a random b. Adversary has negligible advantage in guessing b.
    - A ZK proof is always WI, but not vice-versa
- WI Proofs used as components inside larger protocols
  - Sometimes with certain other useful properties
    - e.g. WI-PoK, "Sigma protocols"
- Defined in standalone setting, but WI property is preserved under "parallel composition"

Non-Interactive ZK

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!
- NIZK: a trusted "common random string" (CRS) is published, and the proof/verification is w.r.t CRS

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!
- NIZK: a trusted "common random string" (CRS) is published, and the proof/verification is w.r.t CRS
  - NIZK property: a simulator can simulate the CRS and the proofs

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!
- NIZK: a trusted "common random string" (CRS) is published, and the proof/verification is w.r.t CRS
  - NIZK property: a simulator can simulate the CRS and the proofs
  - Note: CRS is a part of the proof, but prover is not allowed to choose it (otherwise no soundness)

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!
- NIZK: a trusted "common random string" (CRS) is published, and the proof/verification is w.r.t CRS
  - NIZK property: a simulator can simulate the CRS and the proofs
  - Note: CRS is a part of the proof, but prover is not allowed to choose it (otherwise no soundness)
- NIZK schemes exist for all NP languages (using "enhanced" T-OWP)

Non-Interactive ZK

- Can the prover just give a written proof (no interaction) which any one can verify and can simulate too?
  - No soundness: prover can give the simulated proof!
- NIZK: a trusted "common random string" (CRS) is published, and the proof/verification is w.r.t CRS
  - NIZK property: a simulator can simulate the CRS and the proofs
  - Note: CRS is a part of the proof, but prover is not allowed to choose it (otherwise no soundness)
- NIZK schemes exist for all NP languages (using "enhanced" T-OWP)
  - Also can NIZK-ify some ZK protocols in the RO Model (no CRS)

Today

Today

- Zero-Knowledge Proofs

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition
  - Standalone SIM-security for corrupt verifier (ZK property). Soundness (for corrupt prover) separately

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition
  - Standalone SIM-security for corrupt verifier (ZK property). Soundness (for corrupt prover) separately
- Protocols for Graph 3-colorability (and hence all NP properties) using commitment schemes (in turn using OWP)

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition
  - Standalone SIM-security for corrupt verifier (ZK property). Soundness (for corrupt prover) separately
- Protocols for Graph 3-colorability (and hence all NP properties) using commitment schemes (in turn using OWP)
  - Omitted: ZK for several specific statements

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition
  - Standalone SIM-security for corrupt verifier (ZK property). Soundness (for corrupt prover) separately
- Protocols for Graph 3-colorability (and hence all NP properties) using commitment schemes (in turn using OWP)
  - Omitted: ZK for several specific statements
- Useful in "enforcing" honest (but curious) behavior

Today

- Zero-Knowledge Proofs
  - Interactive Proofs (complete and sound), in which the verifier's view is simulatable given just the statement being proven
- Classical security definition
  - Standalone SIM-security for corrupt verifier (ZK property). Soundness (for corrupt prover) separately
- Protocols for Graph 3-colorability (and hence all NP properties) using commitment schemes (in turn using OWP)
  - Omitted: ZK for several specific statements
- Useful in "enforcing" honest (but curious) behavior
- Some variants (NIZK, WI)