Chapter 10

Interactive Proofs and Zero Knowledge

Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

This field was started by

S. Goldwasser

Turing Award in 2012

S. Micali

C. Rackoff

[Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof-Systems, STOC 1985, SIAM J. Comput. 1989]

László Babai

see also:

[László Babai: Trading Group Theory for Randomness. STOC 1985]

Proofs in mathematics

Two main properties:

- **Completeness**: true statement **have** a proof
- **Soundness**: false statements **don't have** a proof

Question: can we have a **CS version** of this notion?
First idea: extend it by adding "efficient" (**poly-time**) computation.

Recall the definition of **NP**

A language $L \in \{0,1\}^*$ is in **NP** if there exists a **poly-time computable** relation R such that

x – a "**statement**"

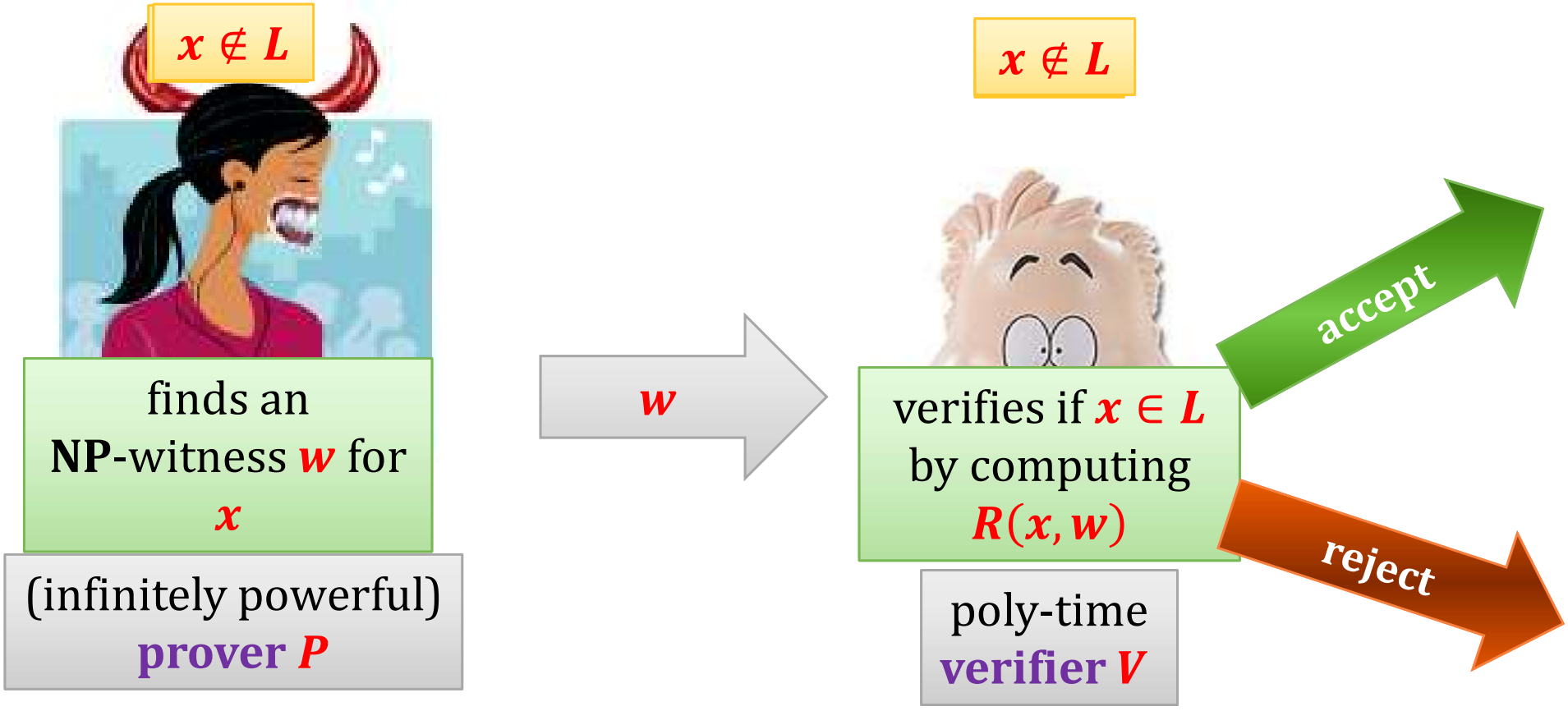
$$L = \{x: \exists_w R(x,w) = \text{true}\}$$

w - an "**NP** witness"

technical assumption: $|w| \leq \text{poly}(|x|)$

One way to look at the NP languages

L – some language in NP defined by a **poly-time computable** relation R



Completeness: If $x \in L$, then the verifier **always accepts**.

Soundness: If $x \notin L$ then the verifier **always rejects** (even if the prover cheats).

How to extend this further?

Add **interaction** and **randomness**

(infinitely powerful)
prover

can **interact**

poly-time
verifier

can "**toss coins**"

Completeness and **soundness** may hold only with some probability.

Notation

Suppose we are given a protocol consisting of two randomized machines P and V.

Machines P and V take some common input x, and then V outputs **yes** or **no**.

We say that (P, V) **accepts** x if V outputs **yes**. Otherwise we say that it **rejects** x.

Interactive proofs

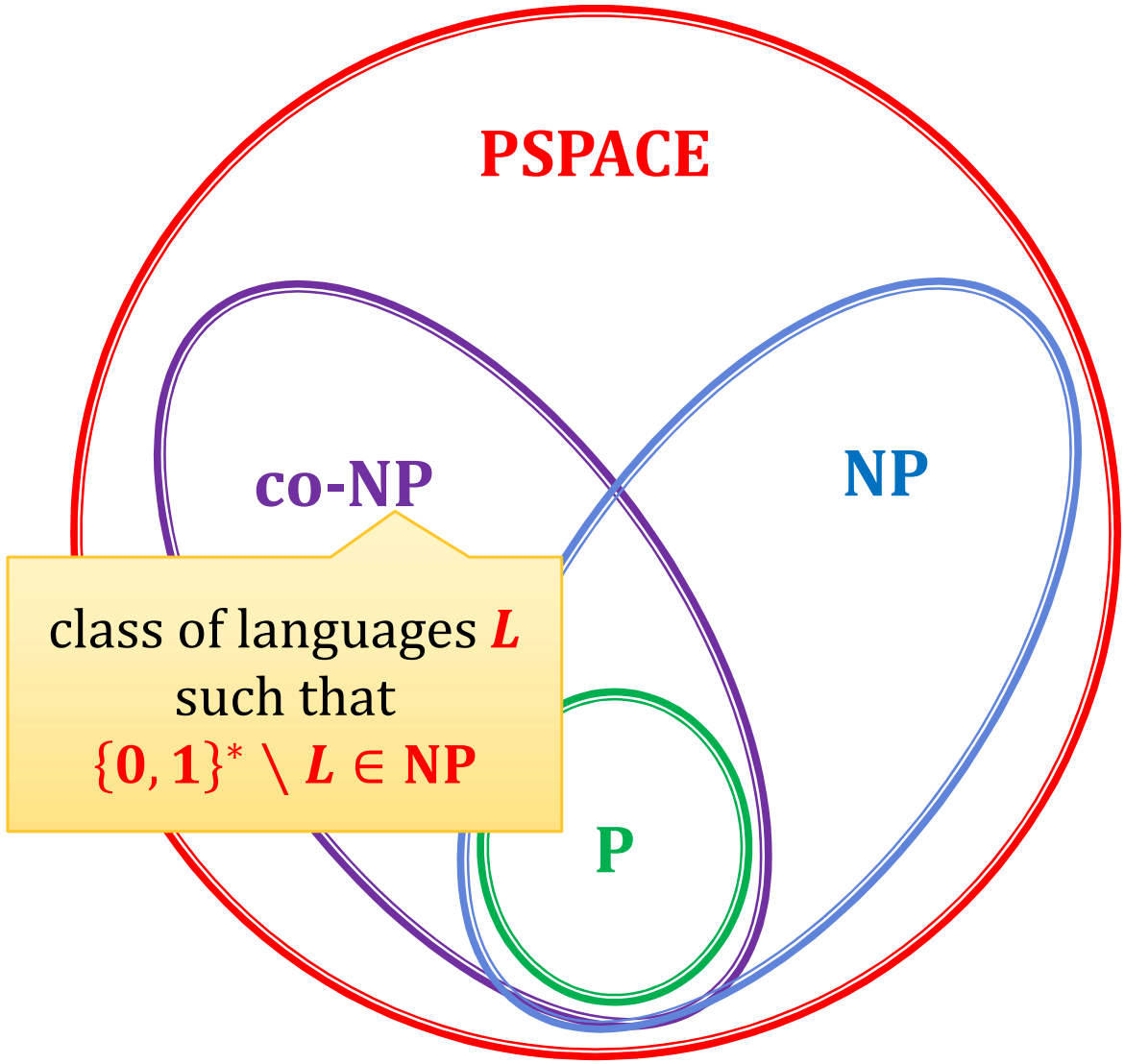
A pair (P, V) is an **interactive proof system** for L if it satisfies the following conditions:

- P has an **infinite computing power** and V is **poly-time**.
- **Completeness**: If $x \in L$, then the probability that (P, V) rejects x is negligible in the length of x.
- **Soundness**: If $x \notin L$ then for **any prover** P^*, the probability that (P^*, V) accepts x is negligible in the length of x.

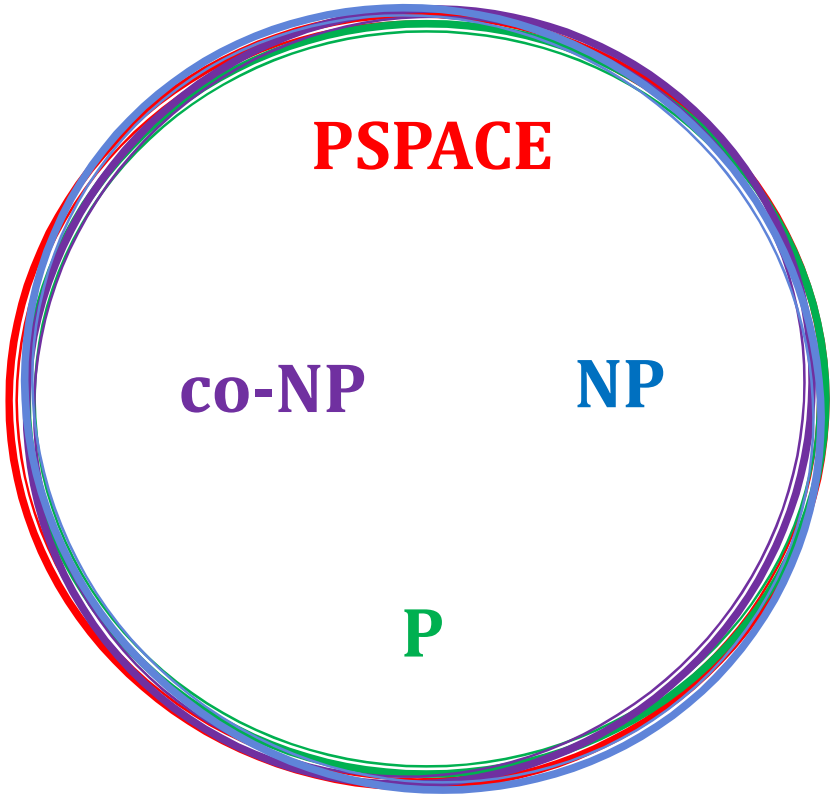
A class of languages that have such proofs is denoted **IP**.

Question: Where does **IP** it belong?

A diagram from complexity theory



in theory it is possible that all these classes are equal:

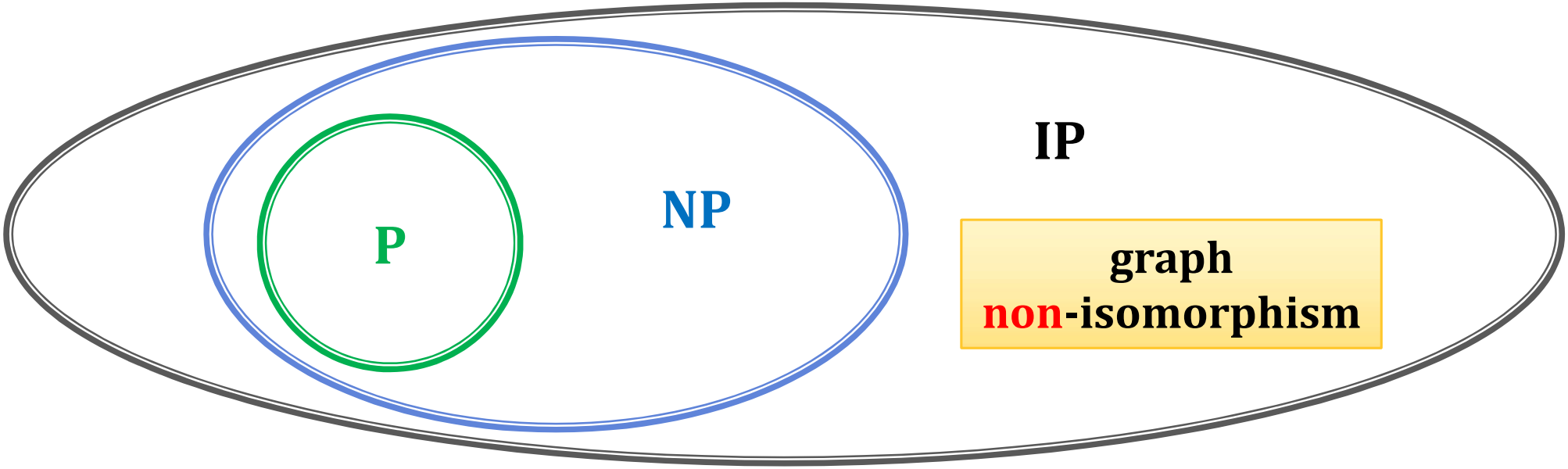


our assumption: they are different

An obvious observation

NP ⊆ IP

(the prover just sends the witness)



We will now show that (probably) **NP** is a proper subset of **IP**.
This will be done by showing an interactive protocol for a **language not known to be in NP**.

Graph isomorphism

A **graph** is a pair (V, E), where E is a binary symmetric relation on V.

A **graph isomorphism between** (V, E) **and** (V', E') is a bijection:

$$\varphi : V \to V'$$

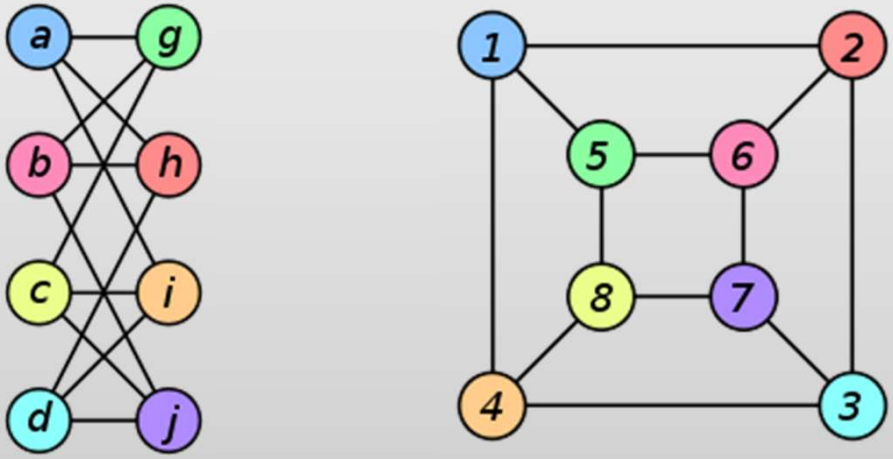
such that

$$(v_1, v_2) \in E \text{ iff } \big(\varphi(v_1), \varphi(v_2)\big) \in E'$$

Graphs G and H are **isomorphic** if there exists an isomorphism between them.

denoted: $G \cong H$

Example



isomorphism:

$f(a) = 1$
$f(b) = 6$
$f(c) = 8$
$f(d) = 3$
$f(g) = 5$
$f(h) = 2$
$f(i) = 4$
$f(j) = 7$

Hardness of graph isomorphism

Without loss of generality, we will consider only isomorphism between (V, E) and (V', E'), where

$$V = V' = \{1, \dots, n\} \text{ (for some } n\text{)}.$$

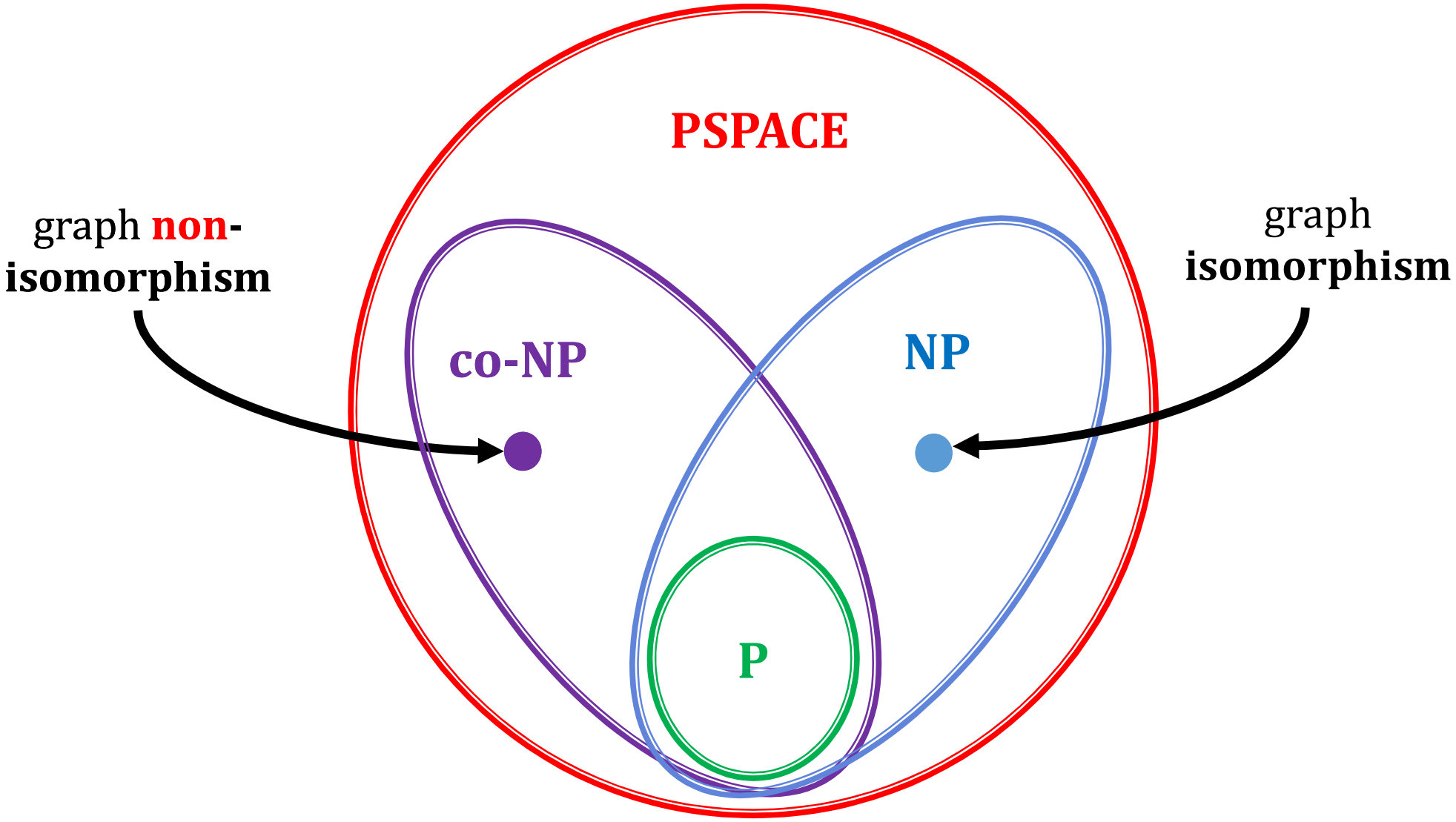
That is, a bijection:

$$\varphi : V \to V'$$

is a permutation of the set $\{1, \dots, n\}$.

No **poly-time algorithm** for the graph isomorphism problem is known.
On the other hand: graph isomorphism is in **NP** (the isomorphism φ is the witness).

On our diagram



Notation

If $G = (V, E)$ is a **graph**, and

π: $V \to V$ is a **permutation**

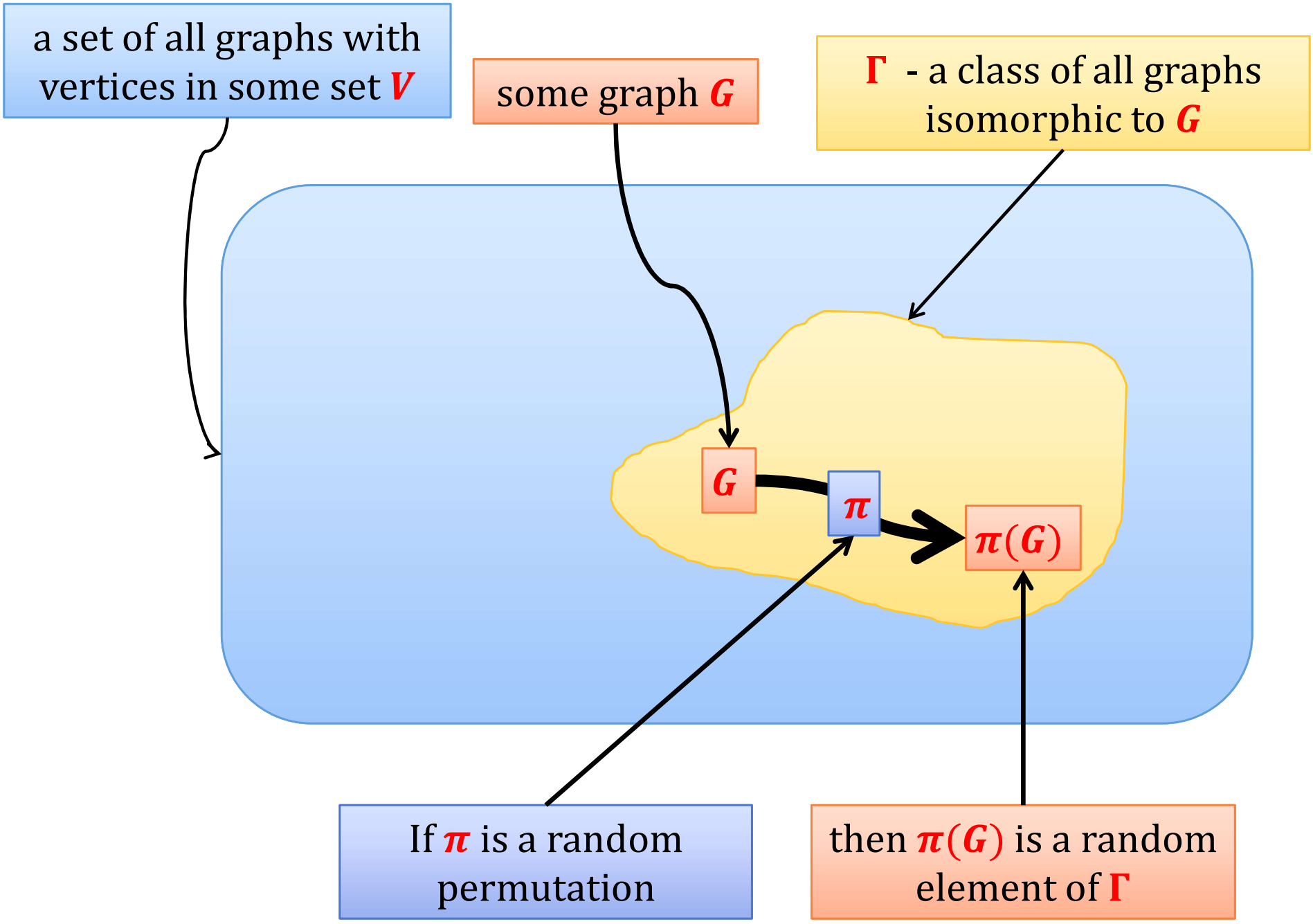
then by $\pi(G)$ we mean a graph

$$G' = (V, E')$$

where

$$(a, b) \in E \quad \text{iff} \quad \big(\pi(a), \pi(b)\big) \in E'$$

A fact

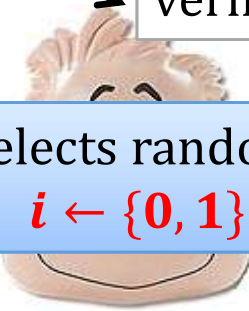

a set of all graphs with vertices in some set V
some graph G
Γ - a class of all graphs isomorphic to G
G
π
$\pi(G)$
If π is a random permutation
then $\pi(G)$ is a random element of Γ

An interactive proof of graph non-isomorphism

statement
graphs G_0 and G_1 are **not** isomorphic

prover

verifier



iterate n times:

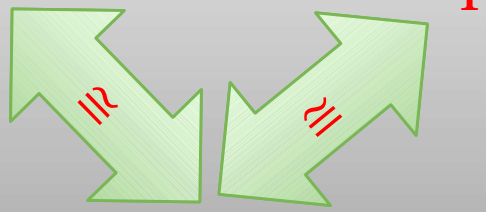
selects random
$i \leftarrow \{0, 1\}$

$H := \pi(G_i)$

checks which
is the case

G_0

G_1



H

sends

$$j := \begin{cases} 0 \text{ if } G_0 \cong H \\ 1 \text{ if } G_1 \cong H \end{cases}$$

accepts if
$i = j$

this technique is called
"**amplification**"

Completeness

Since

G_0 and G_1 are **not** **isomorphic**

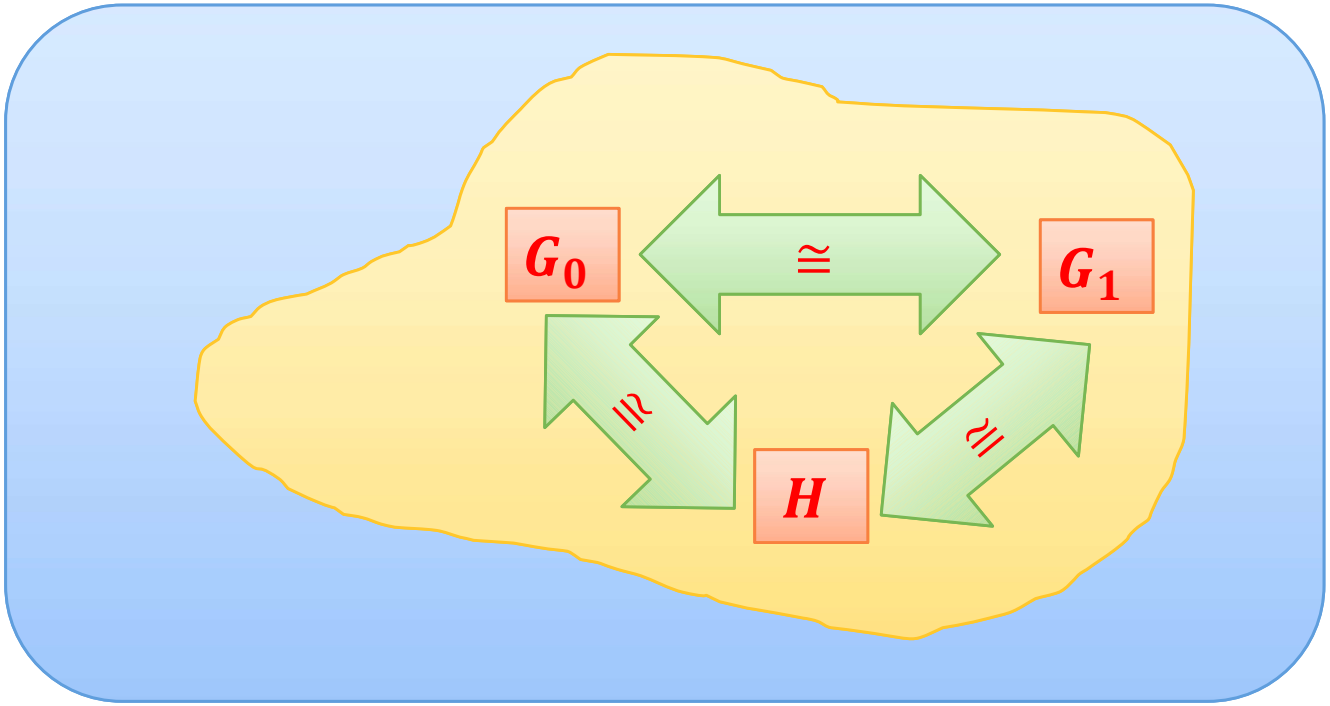
thus, H can be isomorphic with **only one of them**.

Therefore, always $i = j$ and the verifier accepts.

Hence: completeness holds **with probability 1**.

Soundness

Suppose $G_0 \cong G_1$. Then $G_0 \cong H \cong G_1$



So $i = j$ with probability $1/2$.

After n iterations **P(verifier accepts and $G_0 \cong G_1$)** $\leq 2^{-n}$

"soundness error": 2^{-n}

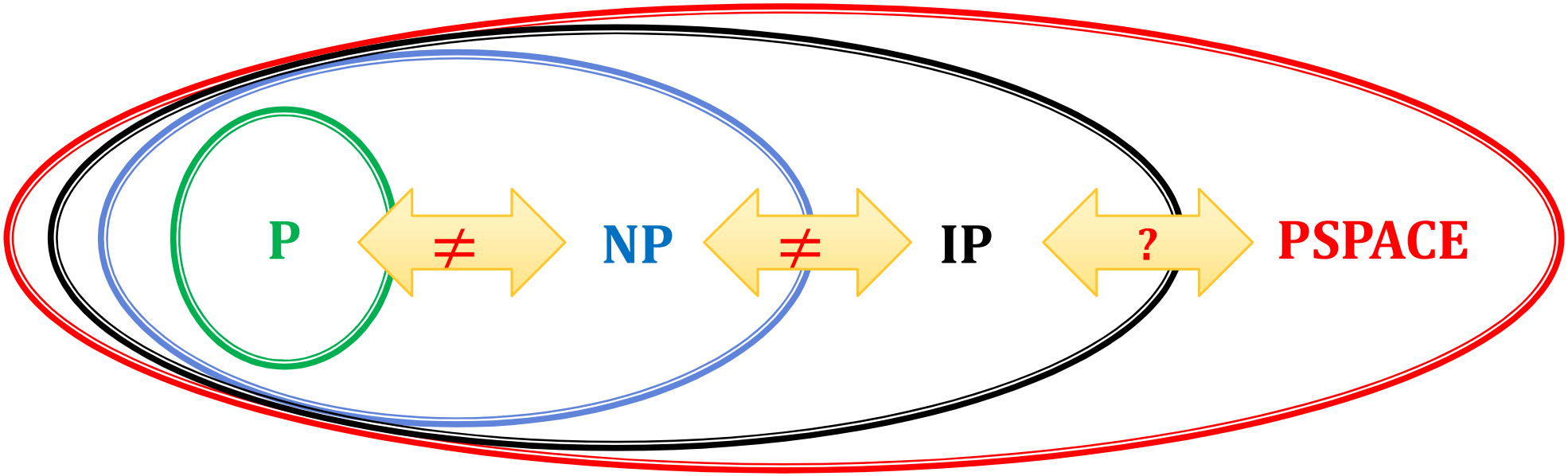
The situation

We have just shown that $\{(G_0, G_1)$: **G_0 are not isomorphic $G_1\} \in$ IP**.

We hence know that (probably) **NP** is a **proper subset of IP**.

On the other hand: it's relatively easy to prove that

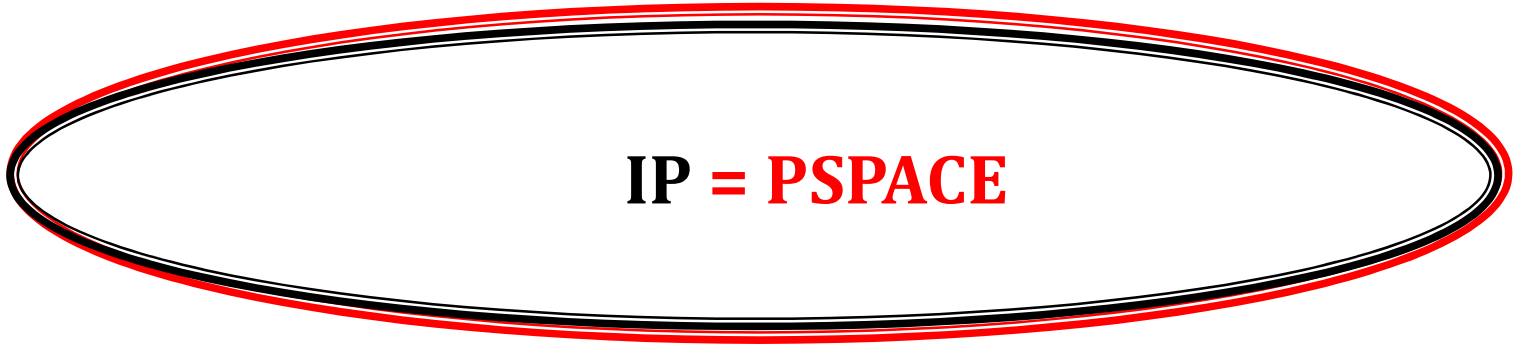
IP ⊆ PSPACE



For a while the other direction ("**PSPACE ⊆ IP**") was an **open problem**.

A celebrated result

PSPACE ⊆ IP



[Adi Shamir "IP=PSPACE", FOCS 1990, J. ACM 39(4): 869-877 (1992)]

(building on work of several other researchers, see **Laszlo Babai "E-mail and the unexpected power of interaction"**)

Main strategy: reduction from the **Quantified Boolean Formula Problem** using an **"arithmetization"** technique.

Tremendous impact on CS

Interactive proofs have **many different variants.**

Most important for **cryptography**:

Zero-Knowledge Protocols

(we will talk about it in a moment).

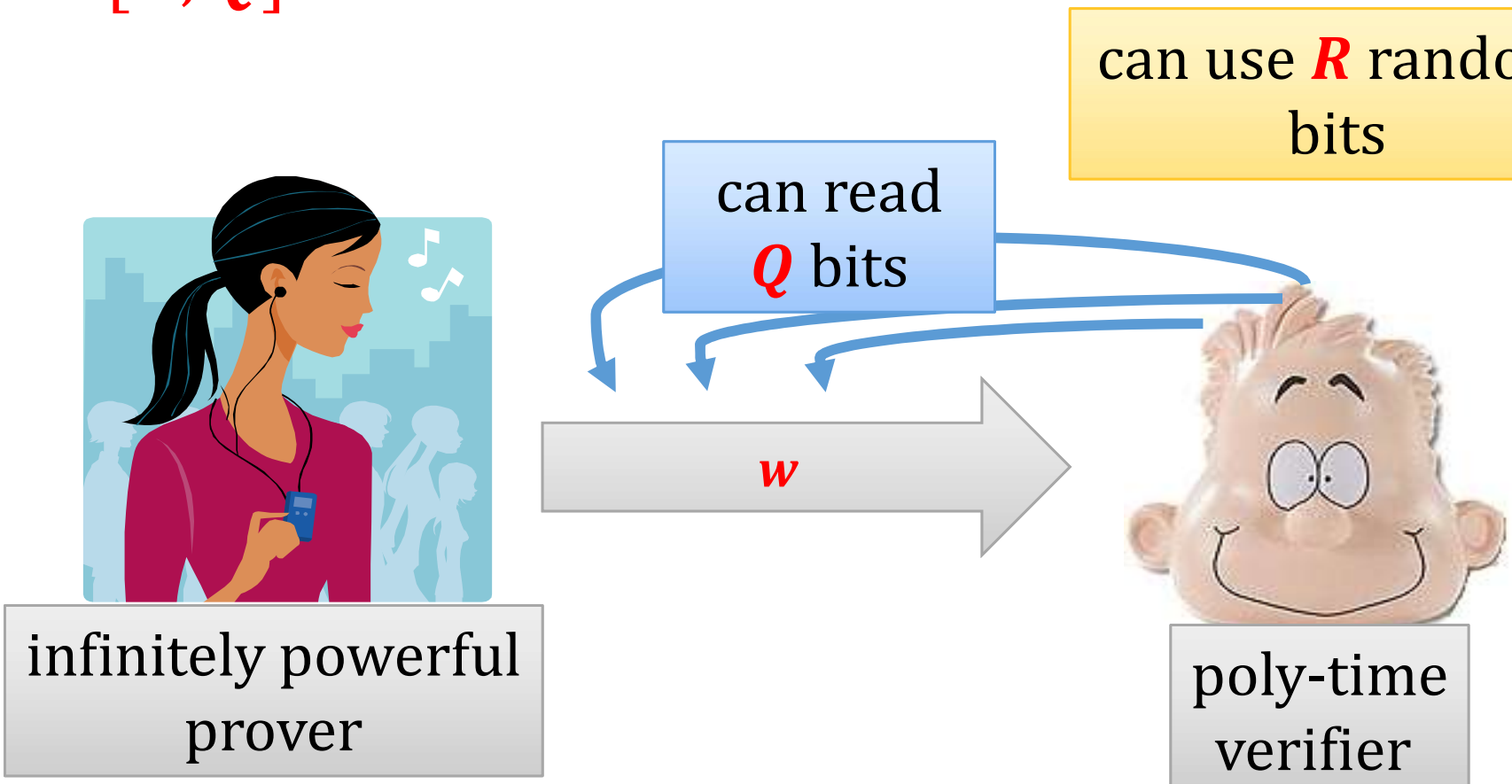
One important generalization:

Probabilistically-Checkable Proofs

(**PCPs**).

Probabilistically-Checkable Proofs

PCP$[R, Q]$:



Soundness error: $\frac{1}{2}$.

Facts about PCP

$$\mathbf{PCP[0,\ 0]\ =\ P}$$

$$\mathbf{PCP[}O\mathbf{(\log(}n\mathbf{)),\ 0]\ =\ P}$$

$$\mathbf{PCP[0,}O\mathbf{(\log(}n\mathbf{))]\ =\ P}$$

$$\mathbf{PCP[0,\ poly(}n\mathbf{)]\ =\ NP}$$

PCP Theorem

$$\mathbf{PCP[}O\mathbf{(\log}\ n\mathbf{),}O\mathbf{(1)]\ =\ NP}$$

Håstad: this can be even equal to **3**

Very deep result with connections to **hardness of approximation algorithms**.

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
    1. introduction and definitions
    2. protocol for graph isomorphism
    3. protocol for quadratic residuosity
    4. protocol for any NP language
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

Adding cryptography to the picture

What about **privacy** of the prover?

In other words: could we have an interactive proof that

$$x \in L$$

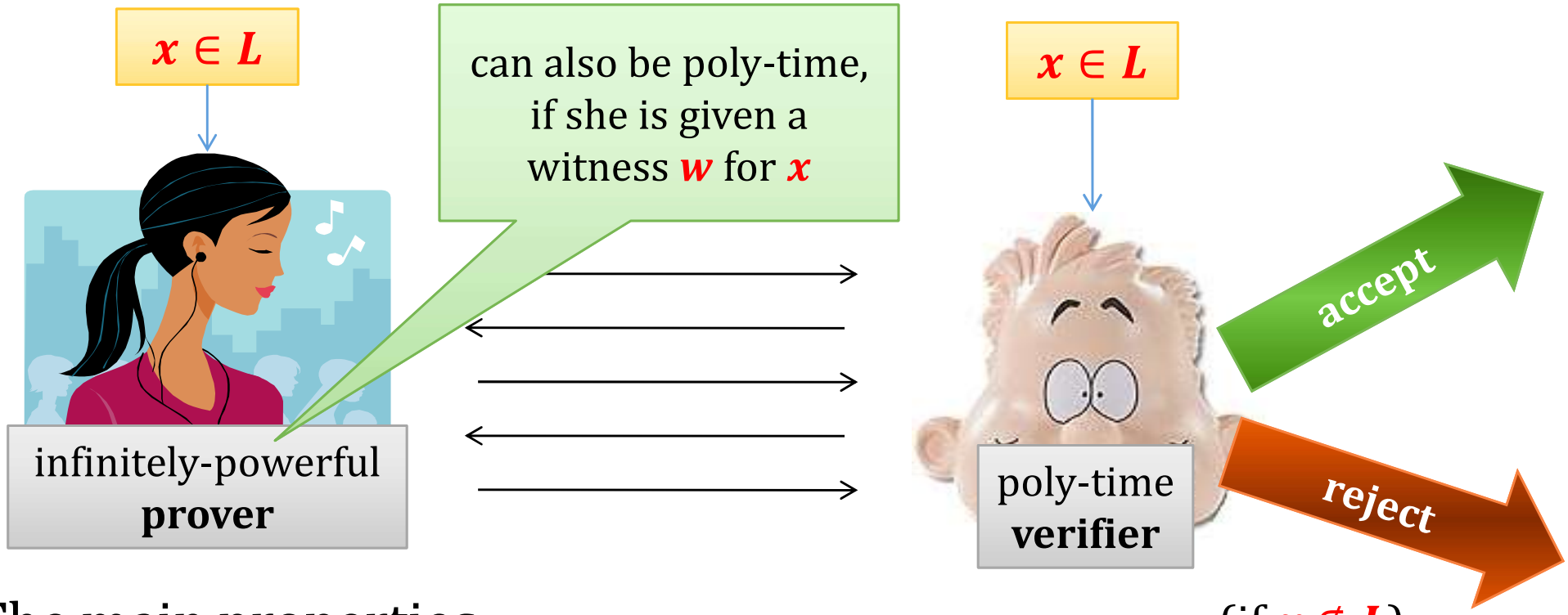
without **revealing any additional information**?

Yes!

This is called **Zero-Knowledge (ZK) Proofs.**

The general picture

$L \in \mathbf{NP}$ – some language (usually not in P)



(if $x \notin L$)

The main properties:

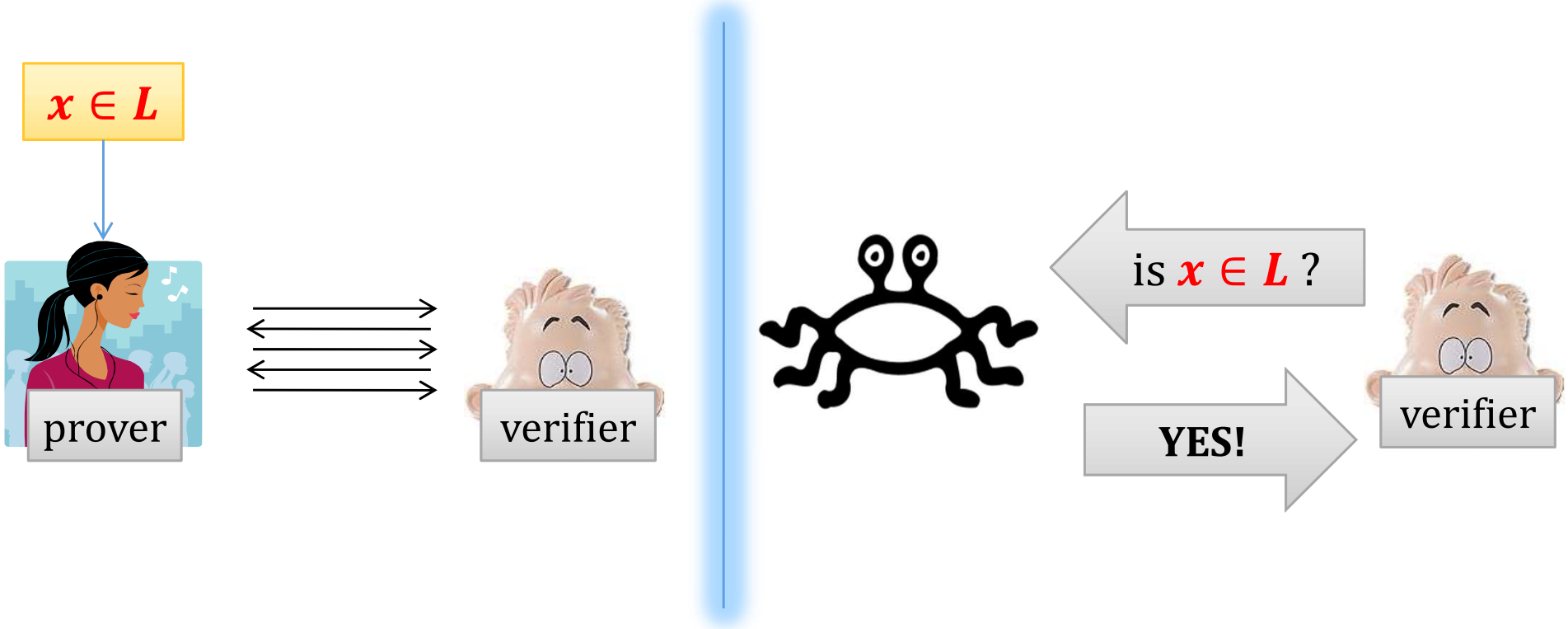
1. **completeness,**
2. **soundness,**

} as before

3. **zero-knowledge** ⟸ new

Zero Knowledge

The only thing that the verifier should learn is that $x \in L$

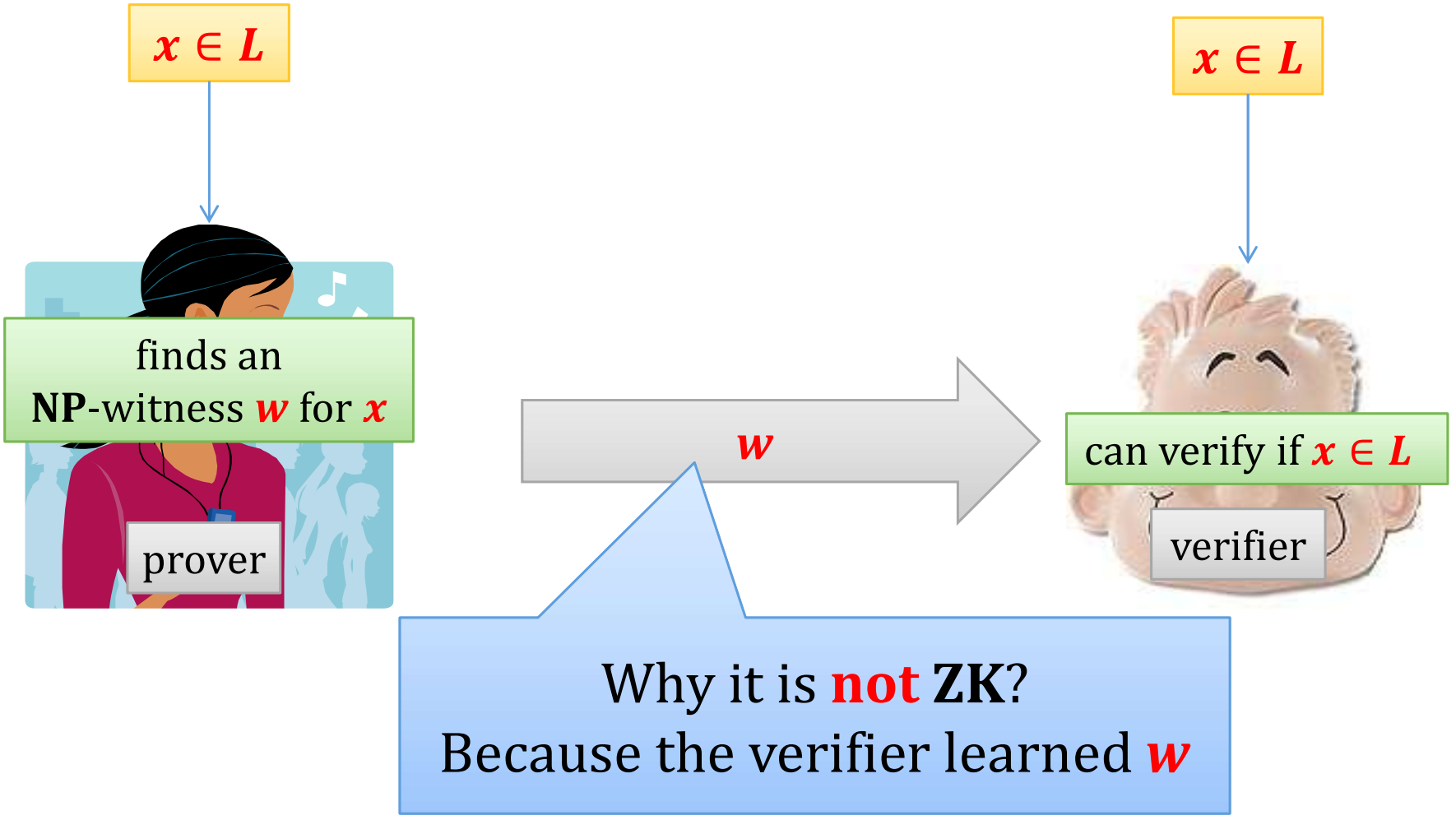


"a cheating V^* should not learn anything besides of the fact that $x \in L$"

(again: we allow some negligible error)

An example of a protocol that is **not** Zero Knowledge

L – some **NP** language



More notation

View$(\boldsymbol{P}, \boldsymbol{V}, \boldsymbol{x})$ – a random variable denoting the "view of $\boldsymbol{V}$",

that is:

1. the random input of $\boldsymbol{V}$ and the input $\boldsymbol{x}$,
2. the **transcript of the communication**.

"a cheating V^* should not learn anything besides of the fact that $x \in L$"

"What a cheating V^* can learn can be simulated without interacting with P"

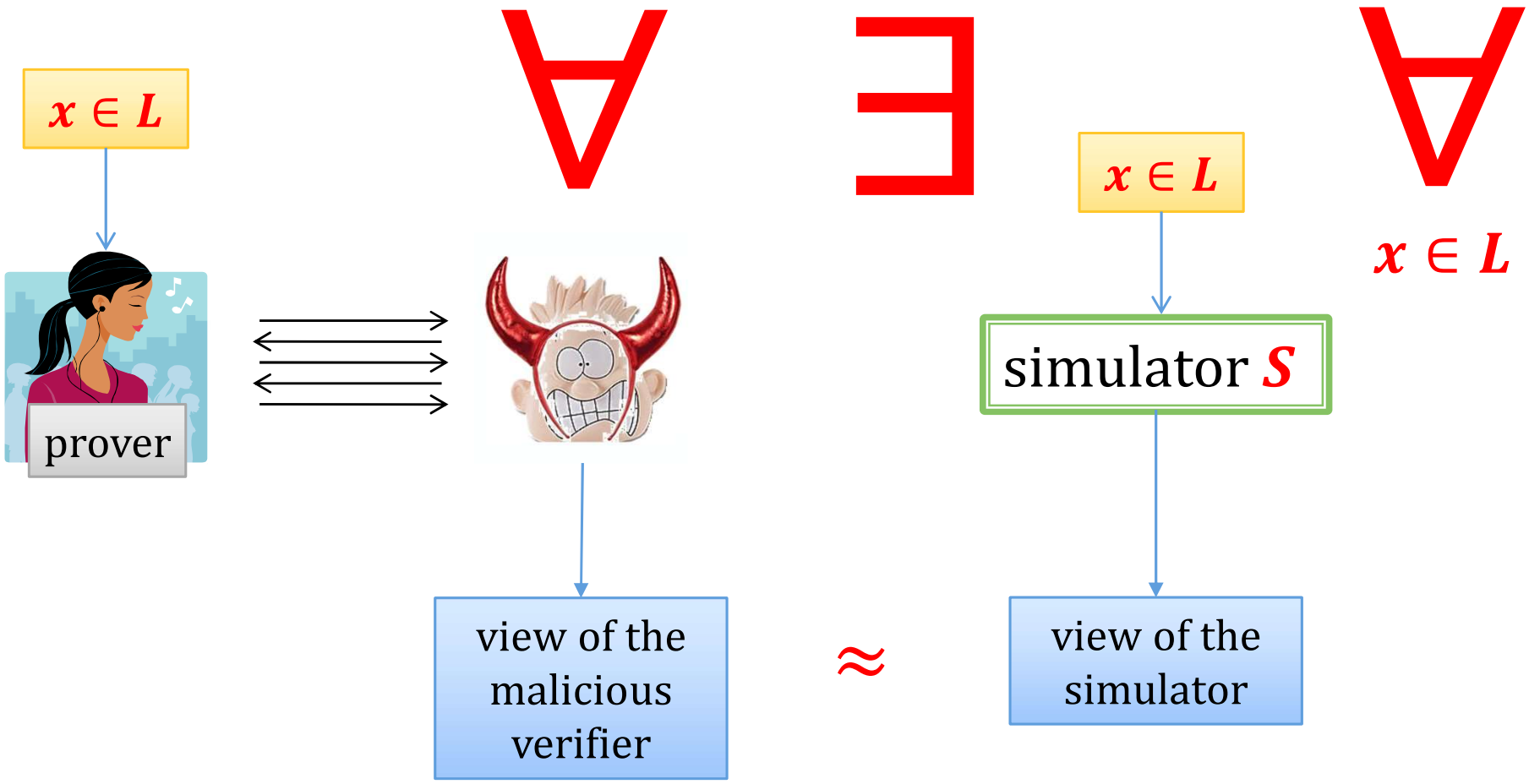
Definition (main idea)

For every (even malicious) poly-time V^* there exists an (expected) poly-time machine S such that

$\{\mathbf{View}(P, V^*, x)\}_{x \in L}$ is "*indistinguishable from*" $\{S(x)\}_{x \in L}$

we will formalize it in a moment

The idea of simulation



Indistinguishability

Let $\alpha = \{A(x)\}_{x \in L}$ and $\beta = \{B(x)\}_{x \in L}$ be two **sets of distributions**.

α and β are **computationally indistinguishable** if for every poly-time D there exists a negligible function ε such that for every $x \in L$

$$\left|P\left(D\left(x, A(x)\right) = 1\right) - P\left(D\left(x, B(x)\right) = 1\right)\right| \leq \varepsilon(|x|) \qquad (*)$$

α and β are **statistically indistinguishable** if $(*)$ holds also for infinitely powerful D.

α and β are **perfectly indistinguishable** if $(*)$ holds also for infinitely powerful D, and $\varepsilon = 0$.

"a cheating V^* should not learn anything besides of the fact that $x \in L$"

Definition (a bit more formally)

For every (even malicious) poly-time V^* there exists an (expected) poly-time machine S such that

$$\{\mathbf{View}(P, V^*, x)\}_{x \in L}$$

is computationally indistinguishable from $\{S(x)\}_{x \in L}$

This is a definition of a **computational** **zero-knowledge**.

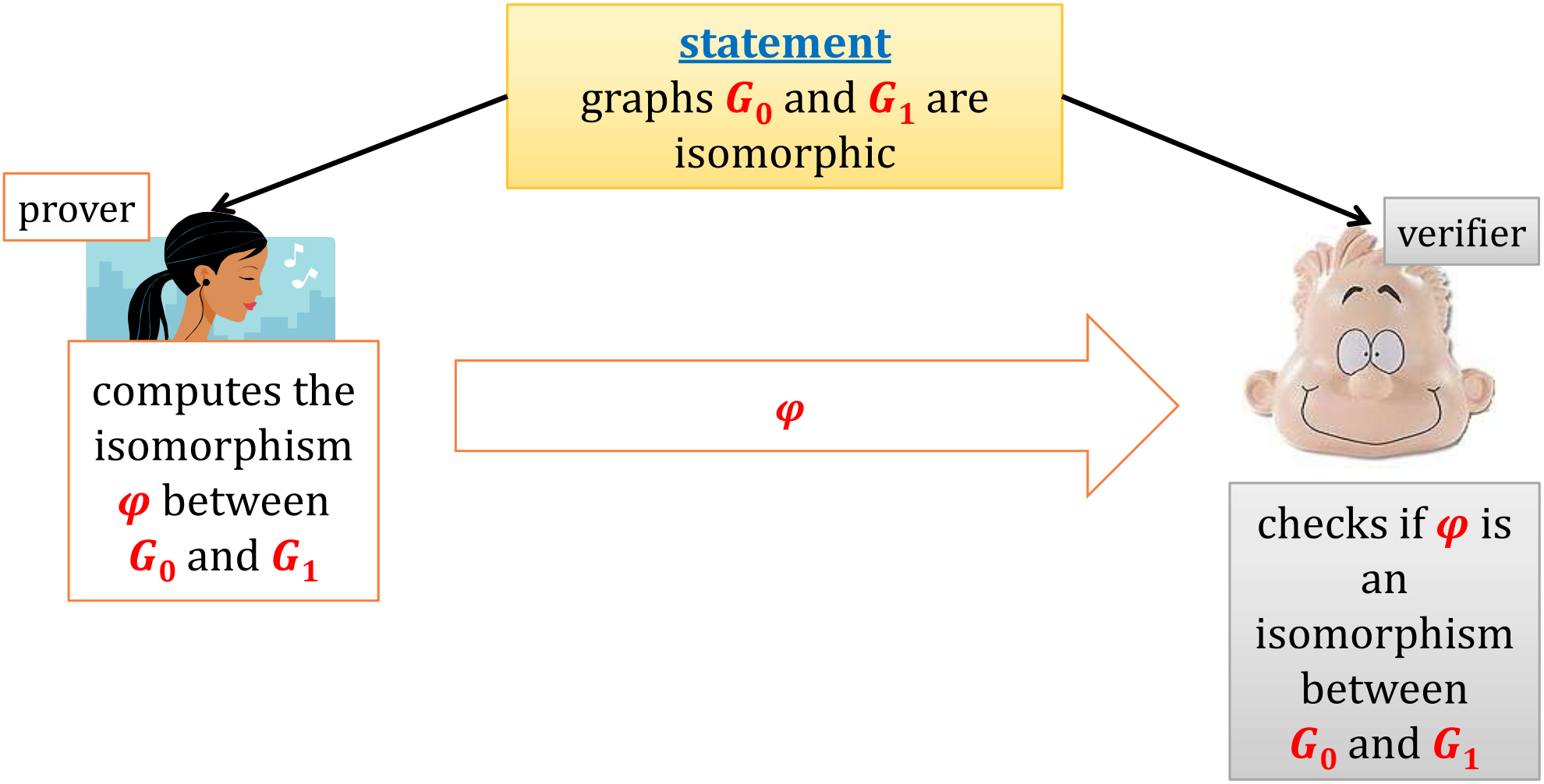
By changing the "**computational** indistinguishability" into

- "**statistical** indistinguishability" we get a **statistical** **zero-knowledge**
- "**perfect** indistinguishability" we get a **perfect** **zero-knowledge**

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
    1. introduction and definitions
    2. protocol for graph isomorphism
    3. protocol for quadratic residuosity
    4. protocol for any NP language
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

A zero-knowledge proof of graph isomorphism – a wrong solution



A zero-knowledge proof of graph isomorphism

statement
graphs G_0 and G_1 are isomorphic

prover

selects a random permutation π

verifier

iterate n times:

$H := \pi(G_1)$

a random $i \in \{0, 1\}$

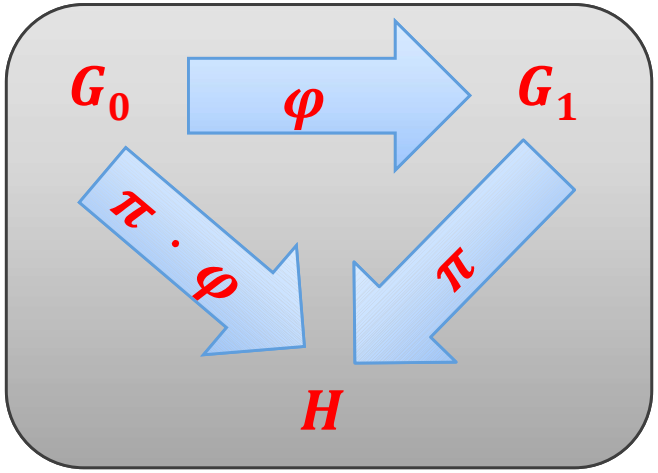
an isomorphism between H and G_i

accepts only if the answer is correct

Note:

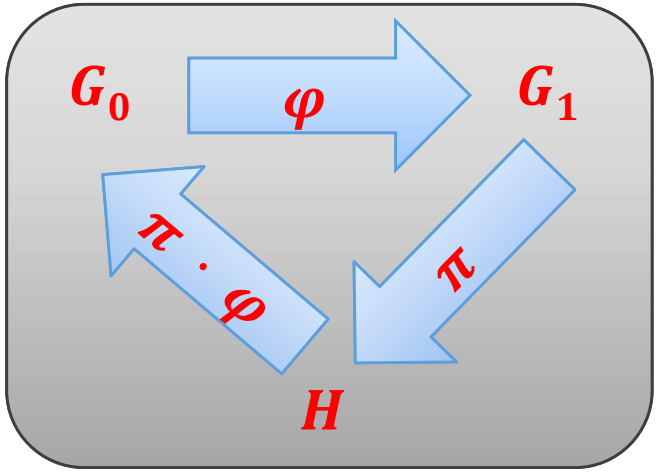
Prover does not need to be infinitely powerful, if he knows the isomorphism isomorphism φ between G_0 and G_i.

- if $i = 1$ then he just sends π
- if $i = 0$ then he sends $\pi \cdot \varphi$



Why is this a zero-knowledge proof system?

- **Completeness**: trivial
- **Soundness**:
  Suppose G_0 and G_1 are **not** isomorphic



Then, **at least <u>one</u> of the following** has to hold:

- G_0 and H are **not** isomorphic
- H and G_1 are **not** isomorphic

→ probability that a verifier rejects is at least $1/2$.

Since the protocol is repeated n times, the probability that the verifier rejects is at least $1 - \left(\frac{1}{2}\right)^n$.

Setting $n := |G_0| + |G_1|$ we are done!

Zero-knowledge?

Intuitively, the zero-knowledge property comes from the fact that:

The only thing that verifier learns is:

- a **permutation** between H and G_0 or G_1

where

- graph H is random graph isomorphic to G_0 (and isomorphic to G_1).

(In fact: we can show that this is a **perfect zero knowledge proof system**.)

More formally

For every poly-time

malicious verifier V^*

there exists an (expected) poly-time

simulator S

such that

$$\{\mathbf{View}(P, V^*, x)\}_{x \in L}$$

is perfectly indistinguishable from $\{S(x)\}_{x \in L}$

statement: graphs G_0 and G_1 are isomorphic

simulator S

G_0 and G_1

select a random permutation π and a bit c

$H := \pi(G_c)$

$i \in \{0, 1\}$

if $i = c$ send an isomorphism between H and G_i

otherwise: restart everything

malicious verifier V^*

output the view of V^*

The running time

First, observe, that the distribution of H doesn't depend on c (since it is uniform in the class of graphs isomorphic with G_0 and G_1)

Therefore the probability that S needs to restart V^* is equal to $1/2$.

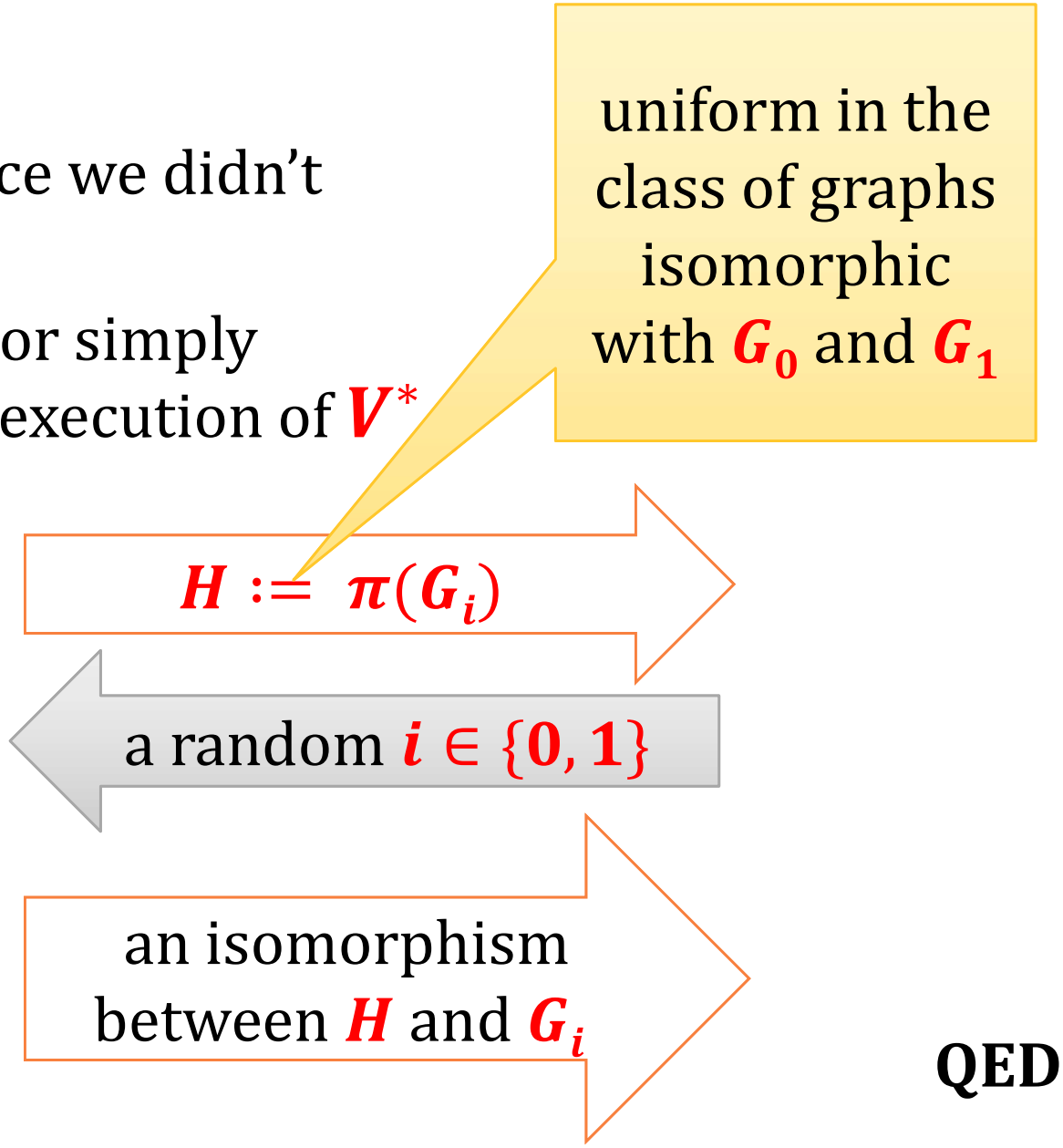
So the expected number of restarts is 2.

Therefore, the running time is (expected) polynomial time.

Indistinguishability of the distributions

Suppose $i = c$, and hence we didn't restart.

In this case, the simulator simply simulated "perfectly" execution of V^* against P.



QED

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
    1. introduction and definitions
    2. protocol for graph isomorphism
    3. protocol for quadratic residuosity
    4. protocol for any NP language
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

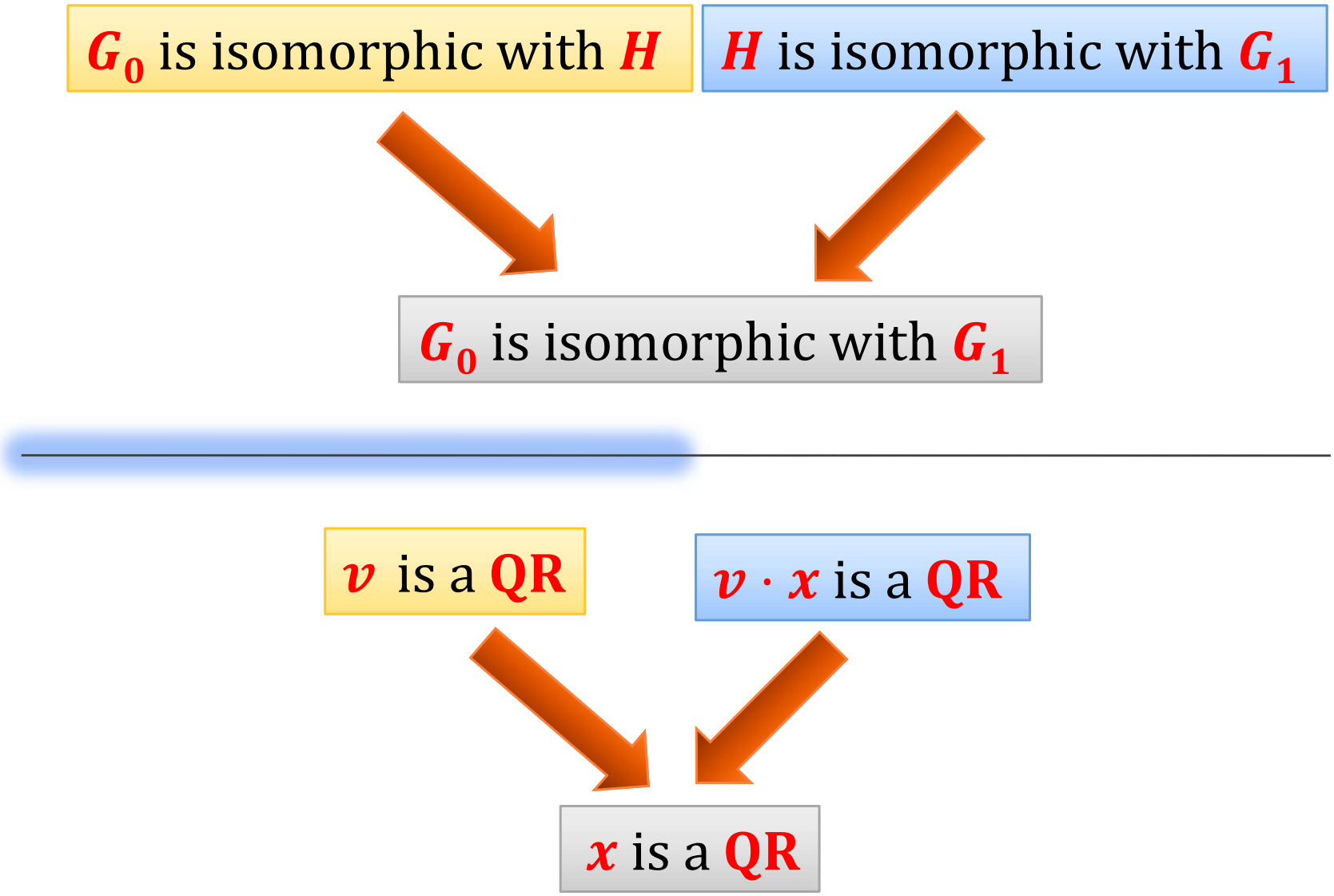
Example

We show a zero-knowledge proof that some x is a quadratic residue modulo N.

How does it work?

Similar to the proof that two graphs are isomorphic!

Main idea


G_0 is isomorphic with H
H is isomorphic with G_1
G_0 is isomorphic with G_1
v is a QR
$v \cdot x$ is a QR
x is a QR

RSA modulus N,
statement: $x \in Z_N^+$ in $\mathbf{QR}_N$

y such that
$y^2 = x \bmod N$

iterate n times:

chose a random
$u \leftarrow Z_N^*$

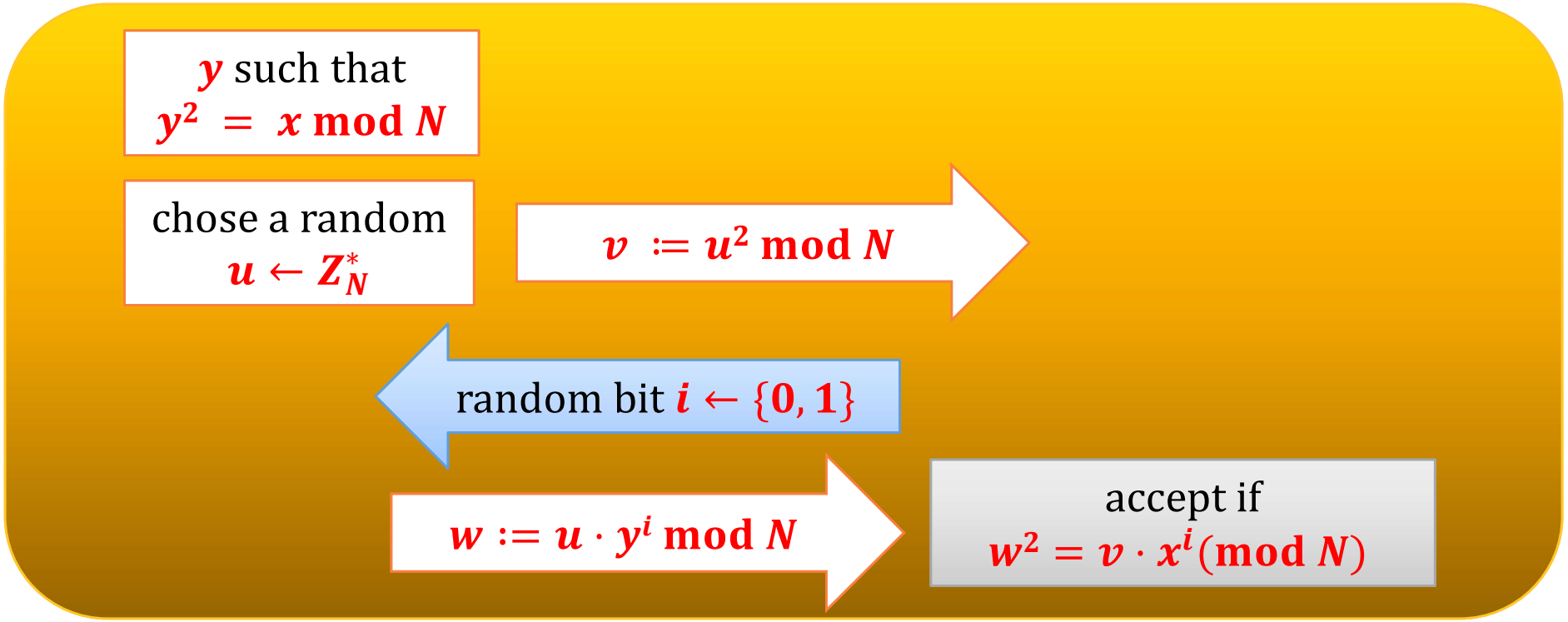
$v := u^2 \bmod N$

random bit $i \leftarrow \{0, 1\}$

$w := u \cdot y^i \bmod N$

$= u$ if $i = 0$
$= u \cdot y$ if $i = 1$

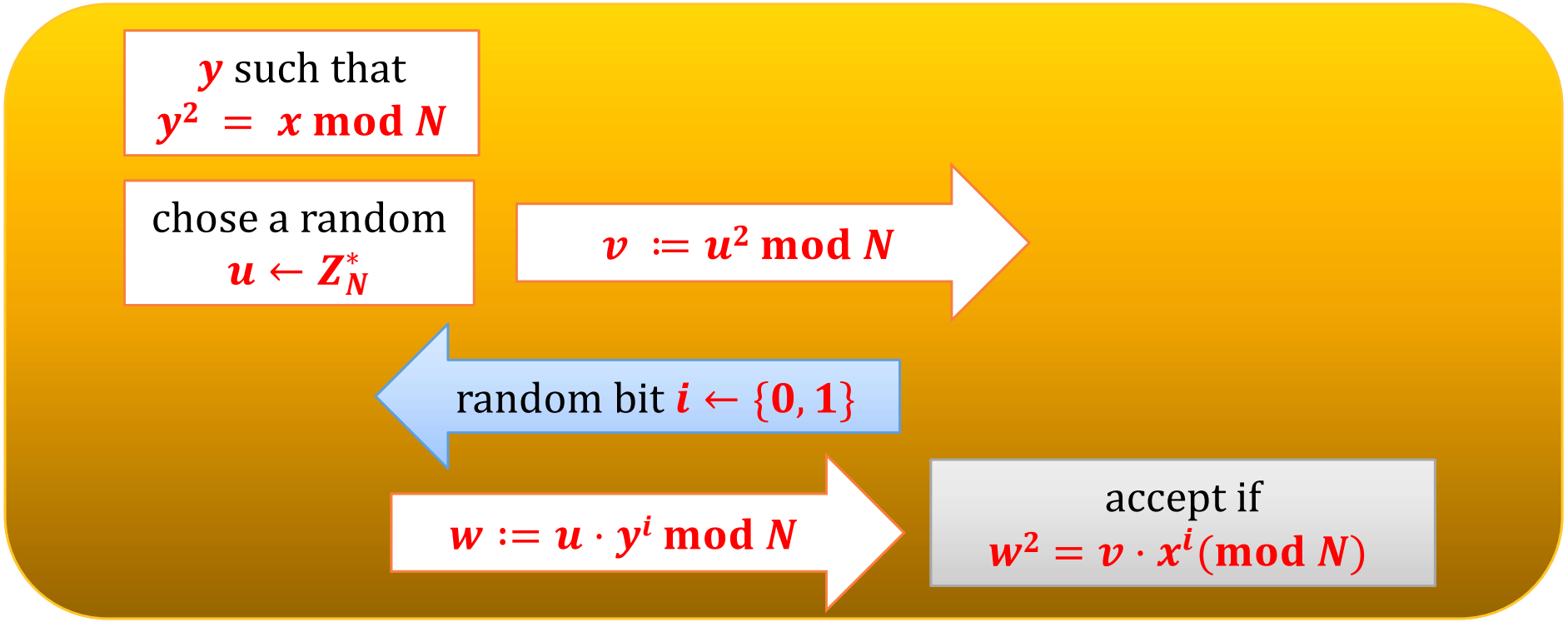
accept if
$w^2 = v \cdot x^i \pmod N$



Why is this a zero-knowledge proof system?

1. Completeness:

$$w^2 = (u \cdot y^i)^2$$
$$= u^2 \cdot (y^2)^i$$
$$= v \cdot x^i$$



2. Soundness: suppose that $x \notin \mathbf{QR}_N$

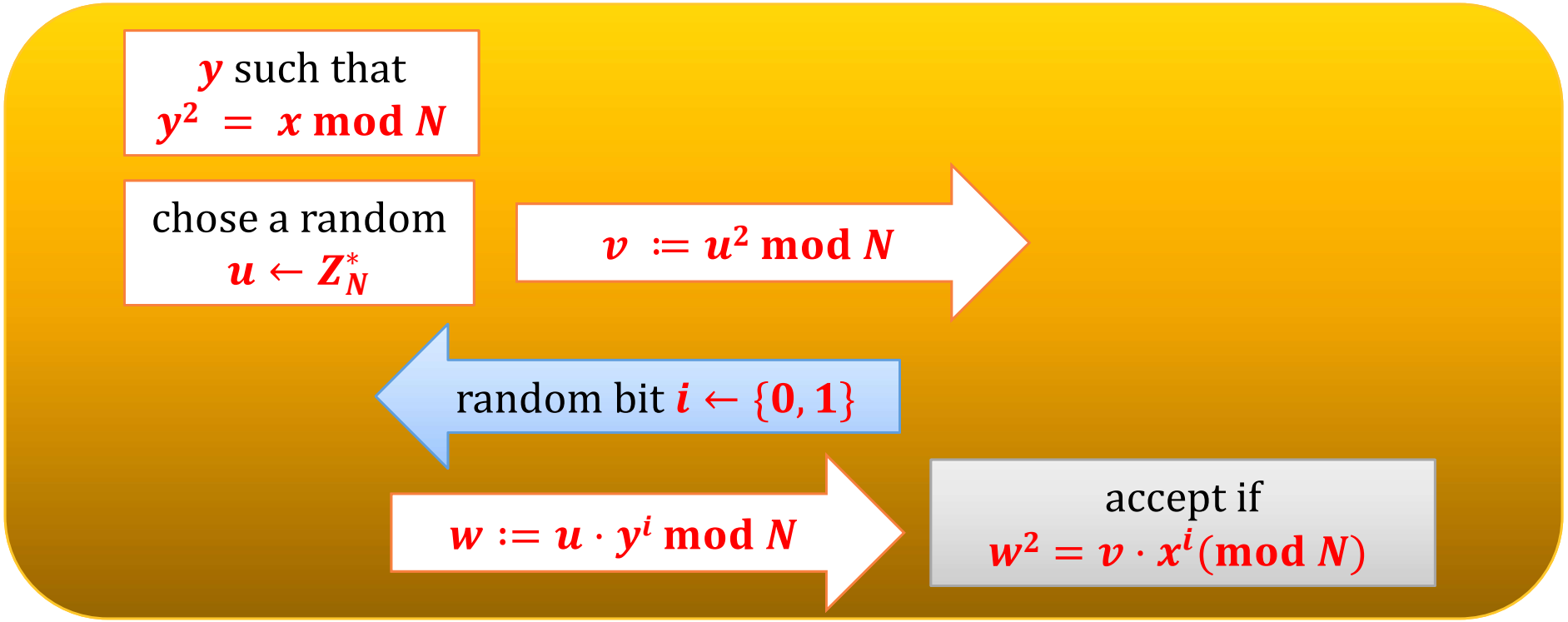
Then

- if v is a $\mathbf{QR}_N$ then the cheating prover gets caught when $i = 1$ since we cannot have

$$\mathbf{QR} \cdot \mathbf{QNR} = \mathbf{QR}$$

- if v is a $\mathbf{QNR}_N$ the cheating prover gets caught when $i = 0$.

So, the prover can cheat with probability at most $1/2$ (in each iteration of the protocol).



3. Zero-knowledge (intuition)

The only information that the verifier gets is:

$v := u^2$

and

- $w := u$ if $i = 0$, or
- $w := u \cdot y$ if $i = 1$.

This obviously gives him no information on y

This also gives him no information on y, since y is "encrypted" with u

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
    1. introduction and definitions
    2. protocol for graph isomorphism
    3. protocol for quadratic residuosity
    4. protocol for any NP language
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

What is provable in NP?

Theorem [Goldreich, Micali, Wigderson, 1986]

Assume that the **one-way functions exist**.

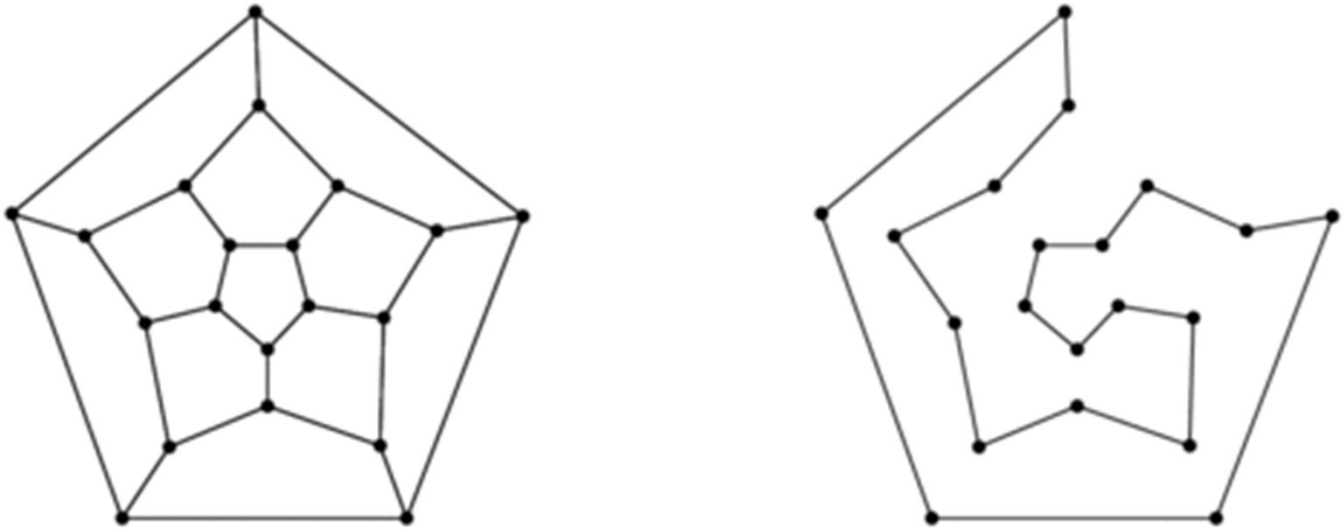
Then, every language $L \in$ **NP** has a computational zero-knowledge proof system.

How to prove it?

It is enough to show it for one **NP-complete** problem!

Take the following NP-complete problem: **Hamiltonian graphs**

Example of a **Hamiltonian cycle**:



Hamiltonian graph – a graph that has a **Hamiltonian cycle**

$$L := \{G : \ G \text{ is Hamiltonian}\}$$

How to construct a ZK proof that a graph G is Hamiltonian?

Of course:

sending the Hamiltonian cycle in a graph G to the verifier doesn't work.

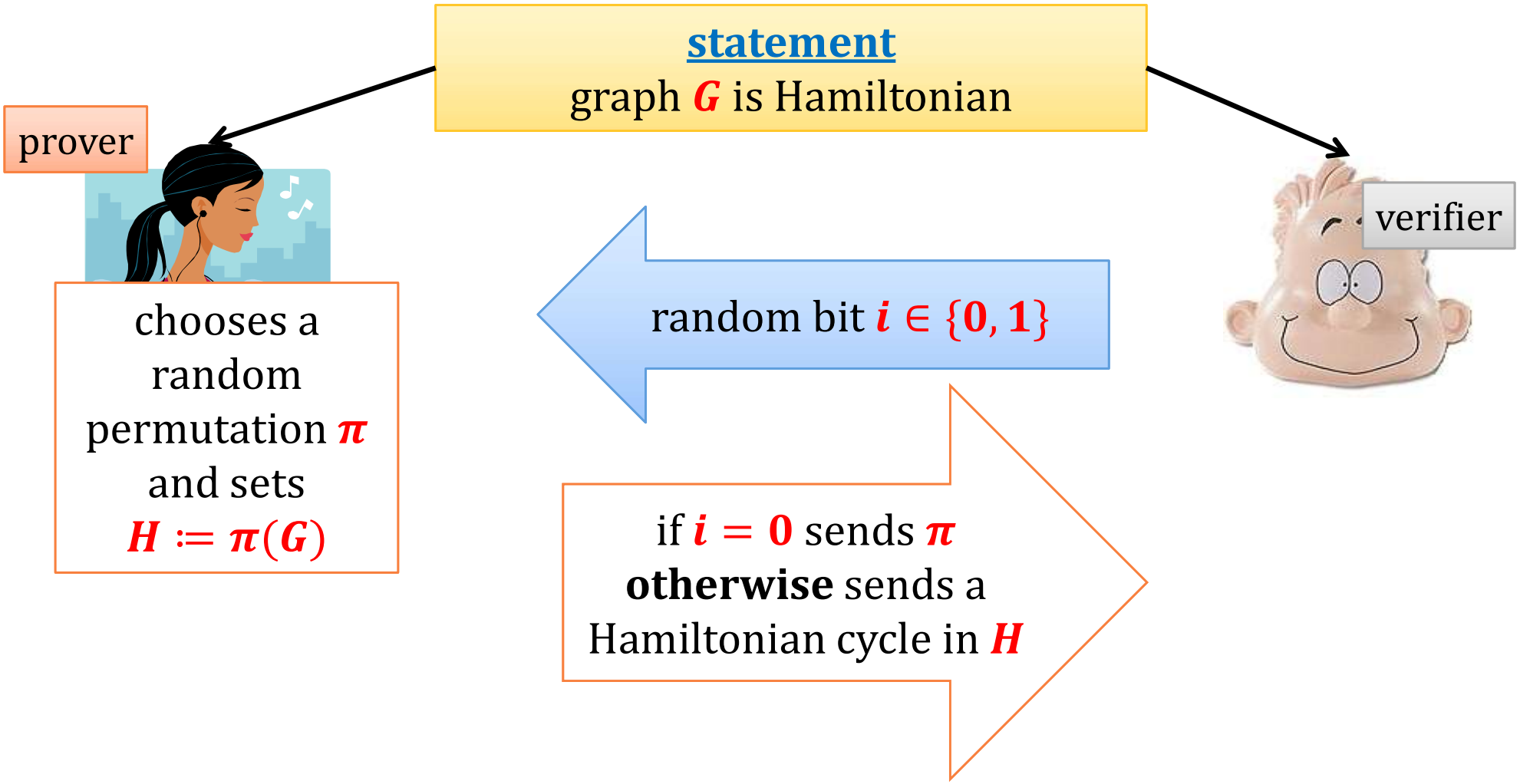
H is Hamiltonian
iff
G is Hamiltonian

Idea:

We permute the graph G randomly – let H be the permuted graph.
Then we prove that

1. H is Hamiltonian,
2. H is a permutation of G.

The first idea:



Problem: Prover can choose his response depending on i.

Solution: use commitments

Remember, that we assumed that the one-way functions exist, so we are "allowed" to use commitments!

Assume the vertices of the graph are natural numbers $\{1, \dots, n\}$.

How to commit to a permutation of a graph?

Represent it as a string

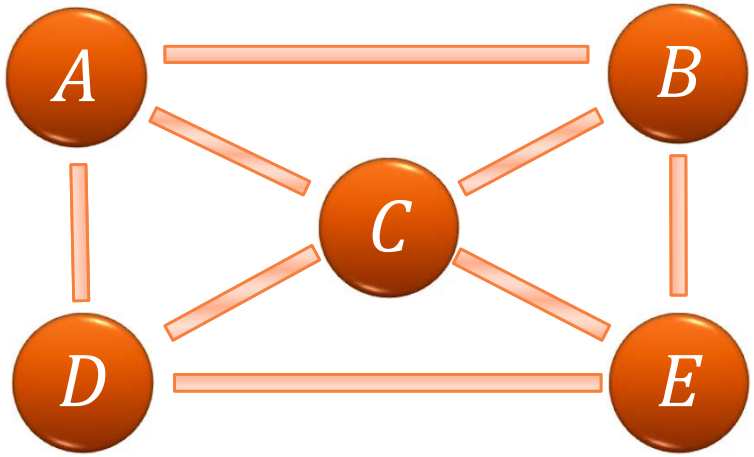
How to commit to a graph?

Represent it as an **adjacency matrix**,
and commit to each bit in the matrix separately.

Example

graph H:

$$M = \{M_{ij}\}_{i,j \in \{A,\ldots,B\}}$$



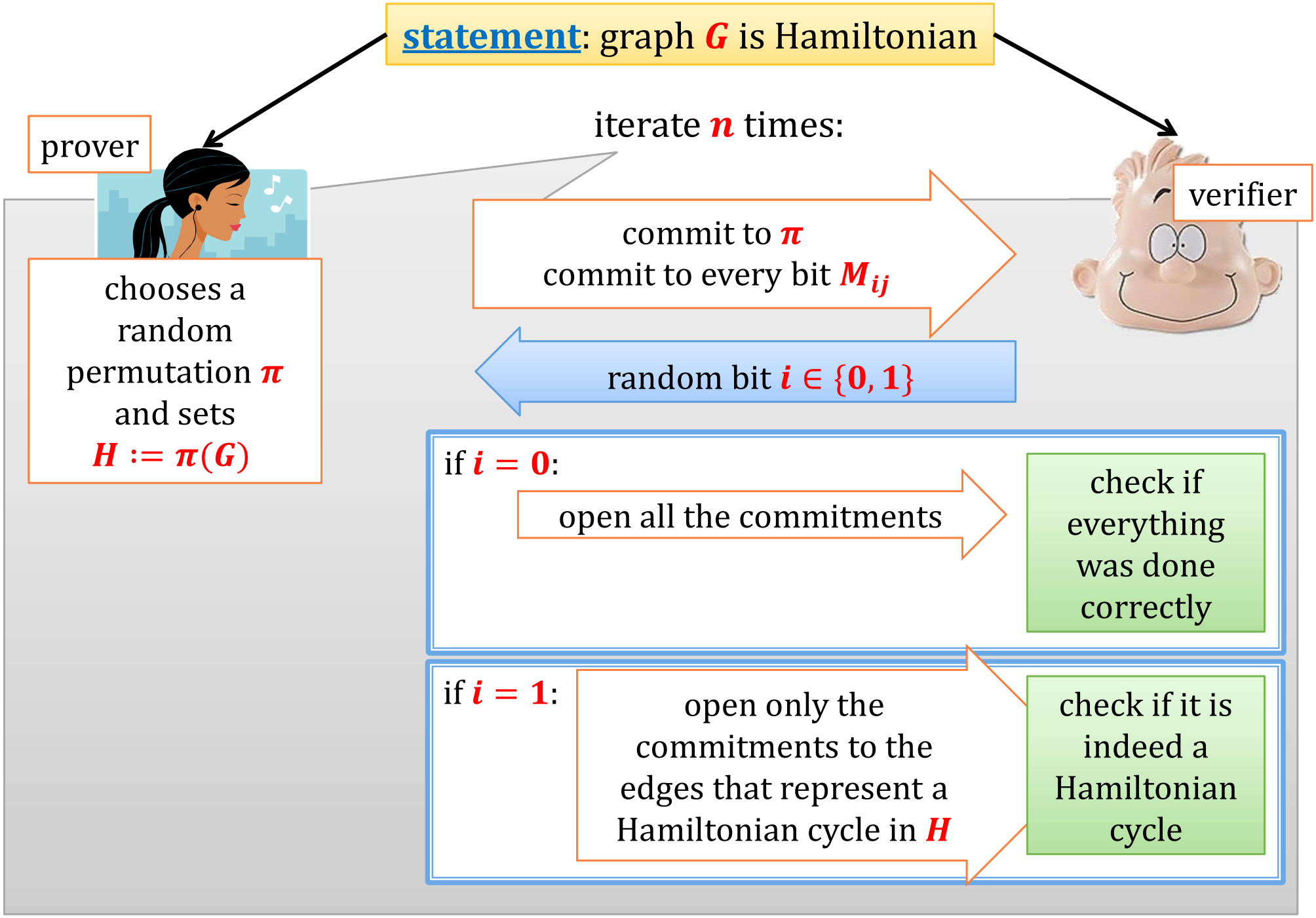
	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	1
C	1	1	0	1	1
D	1	0	1	0	1
E	0	1	1	1	0

to commit to H:

for $i = A, \ldots, E$

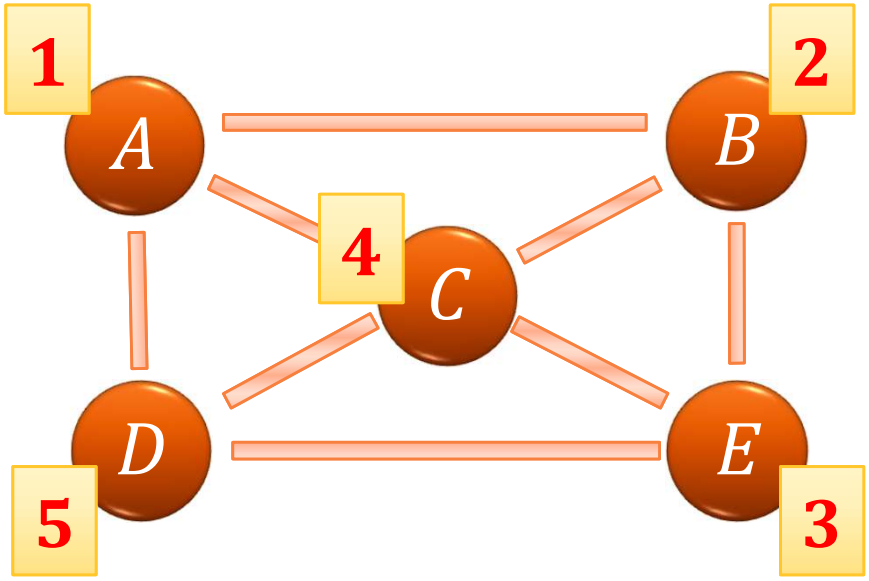
for $j = A, \ldots, E$

Commit(M_{ij})



verifier accepts only if all commitments were open correctly and all checks are ok

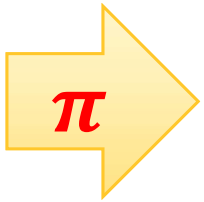
Example of a Hamiltonian graph



		1	2	4	5	3
		A	***B***	***C***	***D***	***E***
1	***A***	0	**1**	**1**	**1**	0
2	***B***	**1**	0	**1**	0	**1**
4	***C***	**1**	**1**	0	**1**	**1**
5	***D***	**1**	0	**1**	0	**1**
3	***E***	0	**1**	**1**	**1**	0

Example of a “permuted graph”

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	1
C	1	1	0	1	1
D	1	0	1	0	1
E	0	1	1	1	0



	$\pi(B)$	$\pi(C)$	$\pi(D)$	$\pi(E)$	$\pi(A)$
$\pi(B)$	0	1	0	1	1
$\pi(C)$	1	0	1	1	1
$\pi(D)$	0	1	0	1	1
$\pi(E)$	1	1	1	0	0
$\pi(A)$	1	1	1	0	0

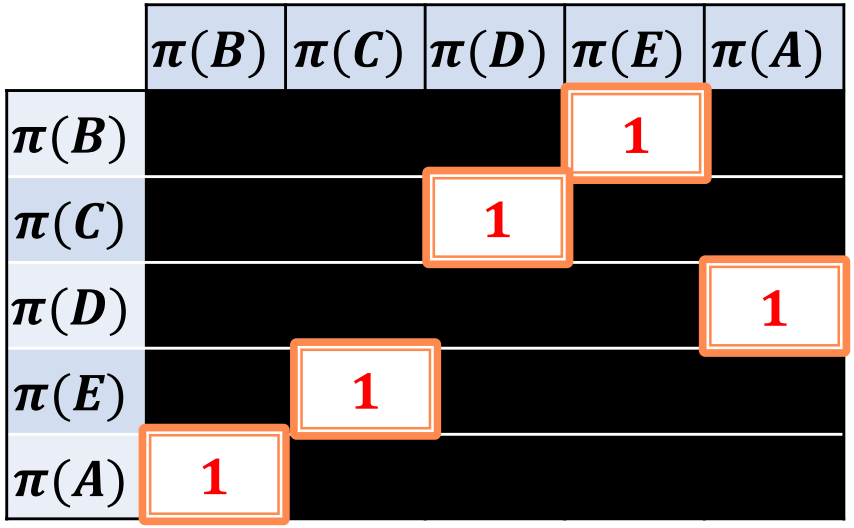
$$\pi(A) = E, \pi(B) = A, \pi(C) = B, \pi(D) = C, \pi(E) = D$$

Case 0:

open everything but **don’t show the Hamiltonian cycle**

Case 1

Open **only** the Hamiltonian cycle



Why is it a ZK proof?

Completeness: trivial

Soundness: If G is not Hamiltonian, then either H is not Hamiltonian or π is not a permutation.

Therefore, to cheat with probability higher than $1/2$ the prover needs to **break the binding property of the commitment scheme**.

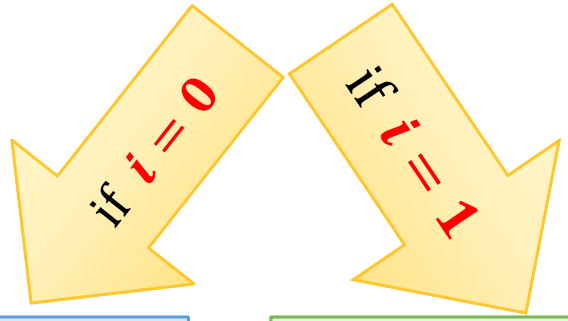
If we use the commitment scheme of **Naor**, this probability is **negligible**, even against an infinitely-powerful adversary

Since the protocol is repeated n times, the probability that the verifier rejects is at least

$$1 - \left(\frac{1}{2}\right)^n.$$

Zero-Knowledge - intuition

"a cheating V should not learn anything besides of the fact that $x \in L$"



P "opens everything", so V just learns a randomly permuted graph G.

P "opens only the Hamiltonian cycle", so V just learns a randomly permuted cycle of vertices

Note, that this gives us only **computational** indistinguishability. This is because the commitment scheme is only computationally binding.

Observation

The honest prover doesn't need to be infinitely powerful, if he receives the **NP**-witness as an additional input!

Corollary

"Everything that is provable is provable in Zero Knowledge!"

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
   1. introduction
   2. Schnorr's protocol for discrete log
4. Non-Interactive Zero-Knowledge
5. Applications

Observation

Recall the In fact, the prover demonstrated not only that x in $\mathbf{QR}_N$, but also that **she knows the square root of x**.

This is called a **Zero-Knowledge Proof of Knowledge**.

It can be defined formally!

Recall the ZK proof for QR

y such that
$y^2 = x \bmod N$

RSA modulus N,
statement: $x \in Z_N^+$ in $\mathbf{QR}_N$

chose a random
$u \leftarrow Z_N^*$

$v := u^2 \bmod N$

random bit $i \leftarrow \{0, 1\}$

$w := u \cdot y^i \bmod N$

accept if
$w^2 = v \cdot x^i \pmod N$

Observation

The prover has to “be ready” to respond to both $\boldsymbol{i := \{0, 1\}}$.

That is: he has to “know”:

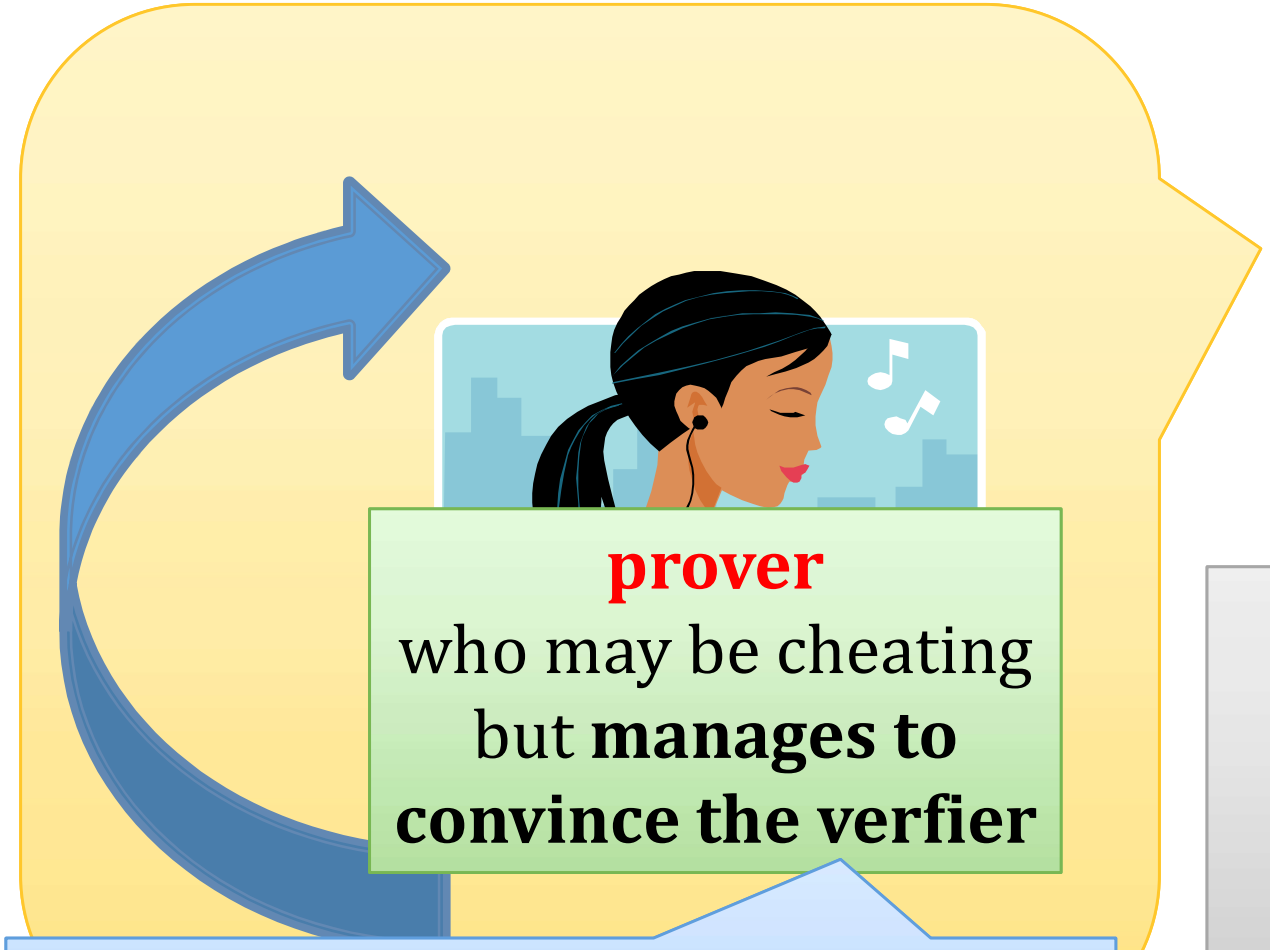
$$\sqrt{v} \text{ and } \sqrt{xv}$$

Hence, he “knows”

$$\frac{\sqrt{xv}}{\sqrt{v}} = \sqrt{x}$$

But what does it mean that a machine “knows some value”?

“Knowledge extractor”



prover
who may be cheating
but **manages to convince the verfier**

a simplification for this lecture:
this happens with probability 1

knowledge extractor
plays the role of the verifier
but **can rewind the prover**

Example for the QR protocol

$v := u^2 \bmod N$

$i := 0$

w

rewind

$i := 1$

$\hat{w}$

knowledge extractor

Note

Since the verifier convinces the prover, we have:

$$w^2 = v$$

and

$$\hat{w}^2 = v \cdot x$$

Thus, the extractor can compute: $y := \frac{\hat{w}}{w}$ and

$$y^2 = x.$$

Therefore he "**extracts**" the knowledge of the witness from the verifier.

Informal definitions

(P, V) is an interactive proof of knowledge for a relation R if it satisfies

- **completeness**

as before

and

- **knowledge-soundness**

there exists a **knowledge extractor** that
from every
prover P^* who convinces the verifier on input x
extracts w such that $R(w, x) = \text{true}$

(P, V) is a zero-knowledge proof of knowledge for a relation R if it is additionally **zero-knowledge**.

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
    1. introduction
    2. Schnorr's protocol for discrete log
4. Non-Interactive Zero-Knowledge
5. Applications

An observation

Proofs of knowledge also make sense for "**trivial languages**".

For example, consider a **cyclic group** G, its **generator** g, and the following statements:

- for $y \in G$ **there exists** x such that $g^x = y$,

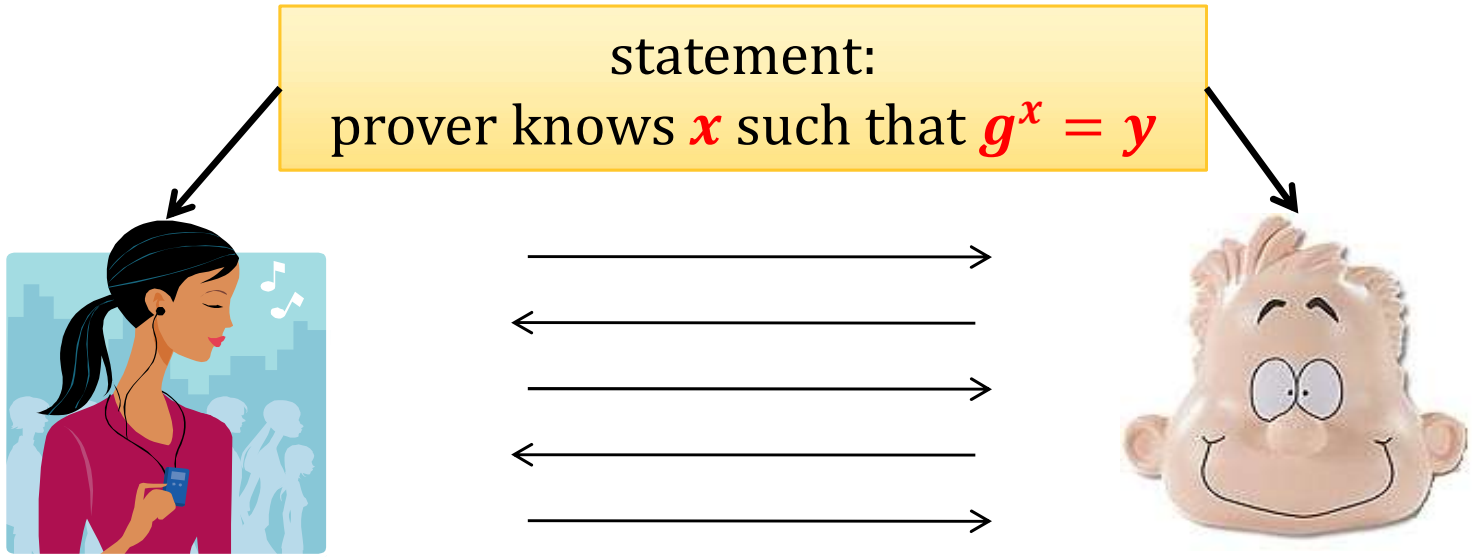
this is trivially **always true**

- for $y \in G$ **verifier knows** x such that $g^x = y$

this may sometimes be false

Schnorr's protocol for discrete log

G – a group with prime order q and generator g.



In other words, the relation is defined as $R := \{(x, y): g^x = y\}$.

The Schnorr's protocol

G – group, $q = |G|$
g – generator

knows x

knows
$y := g^x$

$k \leftarrow Z_q$
$I := g^k$

I

$r \leftarrow Z_q$

r

$s := rx + k \bmod q$

s

output **yes** iff
$g^s \cdot y^{-r} = I$

Fact

verifier has to follow the protocol

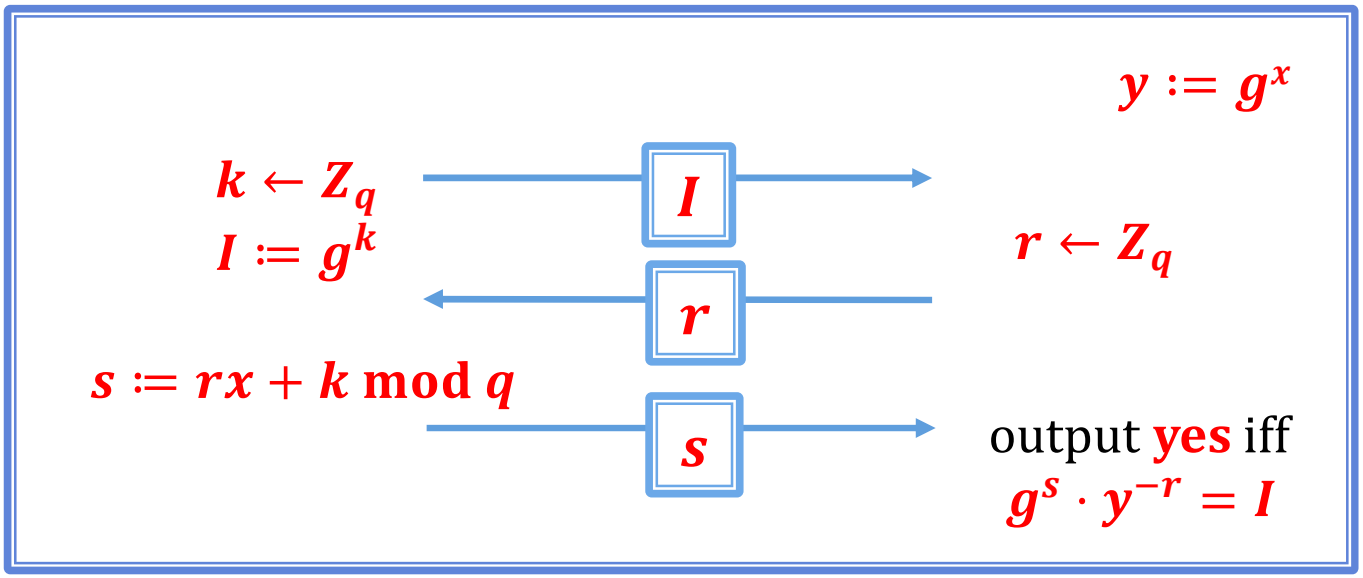
Schnorr's protocol is an **Honest-Verifier Zero-Knowledge Proof of Knowledge** of discrete logarithm.

Schnorr's protocol is **believed** to be also **Zero-Knowledge**, but nobody can prove it (from standard assumptions).

Proof:

We need to show

- completeness
- knowledge-soundness
- honest-verifier zero knowledge



Why is this protocol complete?

$$
\begin{aligned}
g^s \cdot y^{-r} &= g^{rx+k} \cdot (g^x)^{-r} \\
&= g^{rx+k} \cdot g^{-rx} \\
&= g^k \\
&= I
\end{aligned}
$$

Knowledge soundness

I

r_0

s_0

rewind

r_1

s_1

$r_0 \leftarrow Z_q$

knows s_0 such that $g^{s_0} \cdot y^{-r_0} = I$

$r_1 \leftarrow Z_q$

knows s_1 such that $g^{s_1} \cdot y^{-r_1} = I$

We now have

$$g^{s_0} \cdot y^{-r_0} = I = g^{s_1} \cdot y^{-r_1}.$$

Hence

$$y^{r_1 - r_0} = g^{s_1 - s_0}$$

so

$$y = g^{\frac{s_1 - s_0}{r_1 - r_0}}$$

witness
$x = \log_g y$

note: with overwhelming probability
$r_0 \neq r_1$

So, we **extracted the witness**!

Note: in **the full proof** we would also need to consider the case when the malicious verifier succeeds with **probability** < 1.

For ZK: look at the verifier's views:

$$(I, r, s)$$

where

- $I = g^k$ where $k \leftarrow Z_q$
- $r \leftarrow Z_q$
- $s := rx + k \bmod q$

We now show how to simulate them without knowing x.

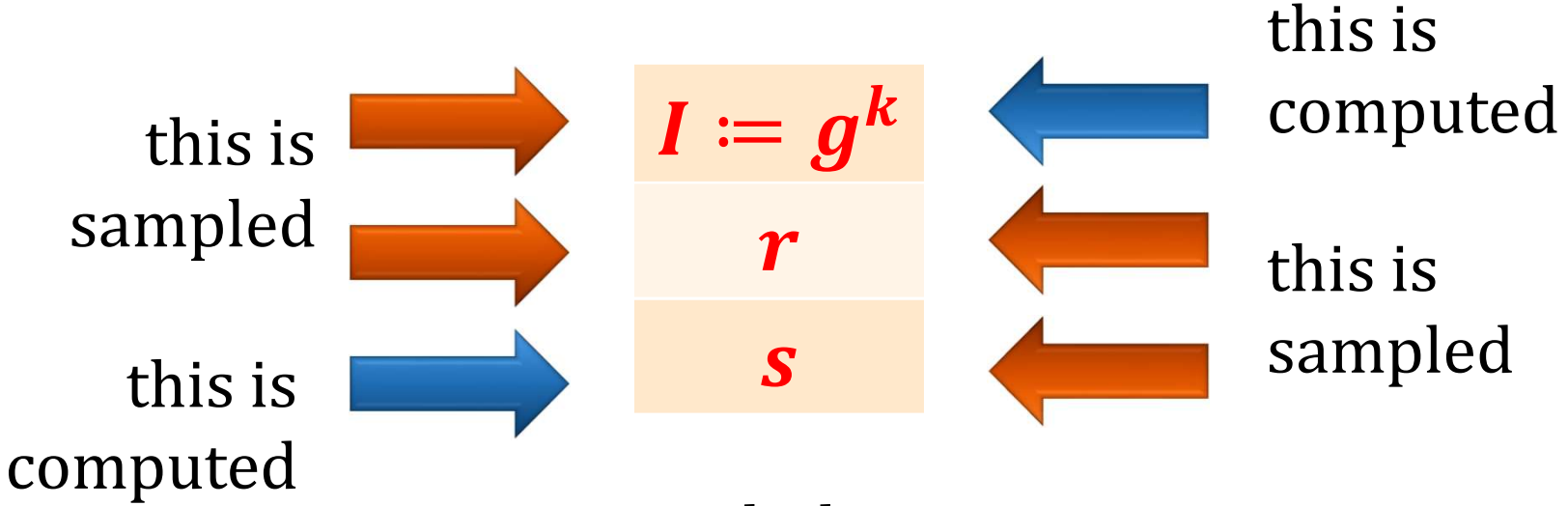
How to simulate?

- **first** sample $r, s \leftarrow Z_q$ and
- **then** compute I as

$$I = g^s \cdot y^{-r}$$

note: this works only because r cannot depend on I ("honest verifier")

Why is the distribution the same?



such that

$$s := rx + k \bmod q$$

It's the same!

Note the difference

The simulator **can** produce tuples (I, r, s) with the right distribution

if he "**starts from** (r, s)"

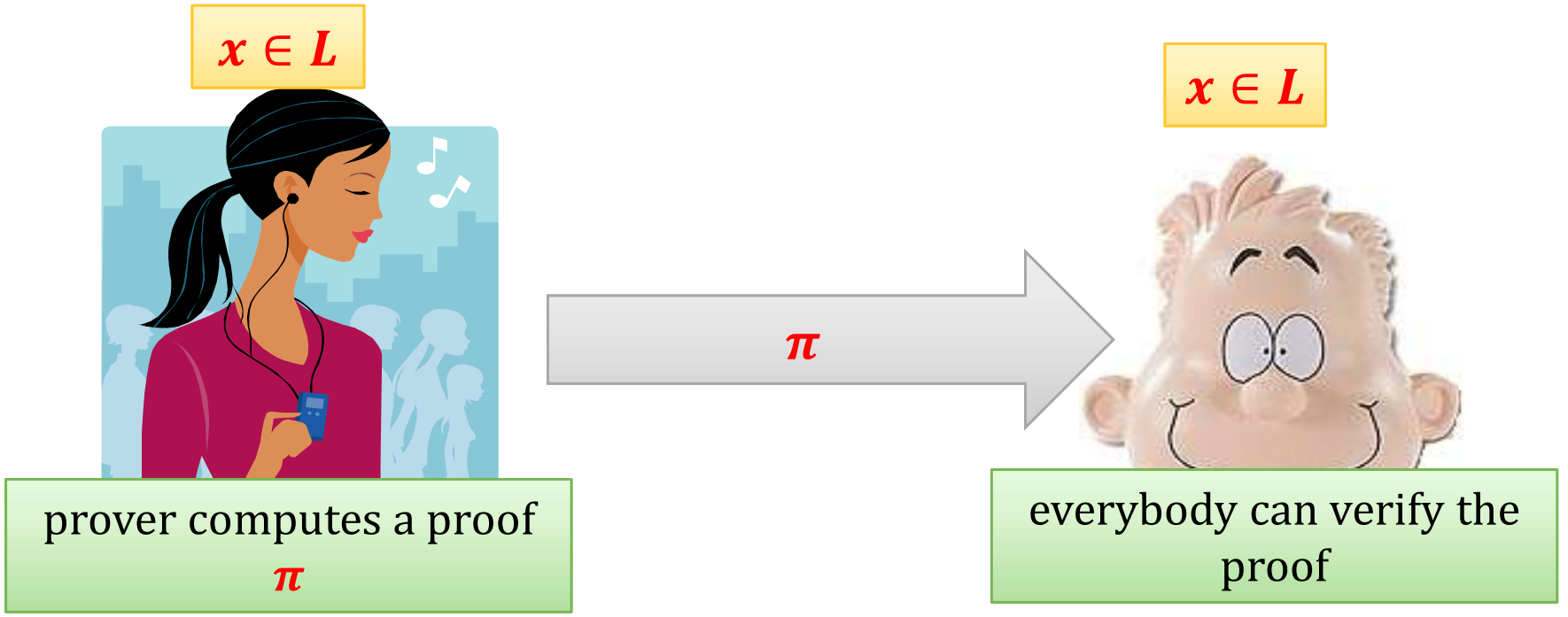
but he **cannot do it**

if he has to "**start from** I" and sampling r is out of his control.

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
    1. introduction
    2. signature schemes from ZK Proofs of Knowledge
5. Applications

Non-interactive zero-knowledge proofs (NIZKs)



Note: we will work in non-standard models, so π is not an **NP-witness**.

How to construct NIZKs

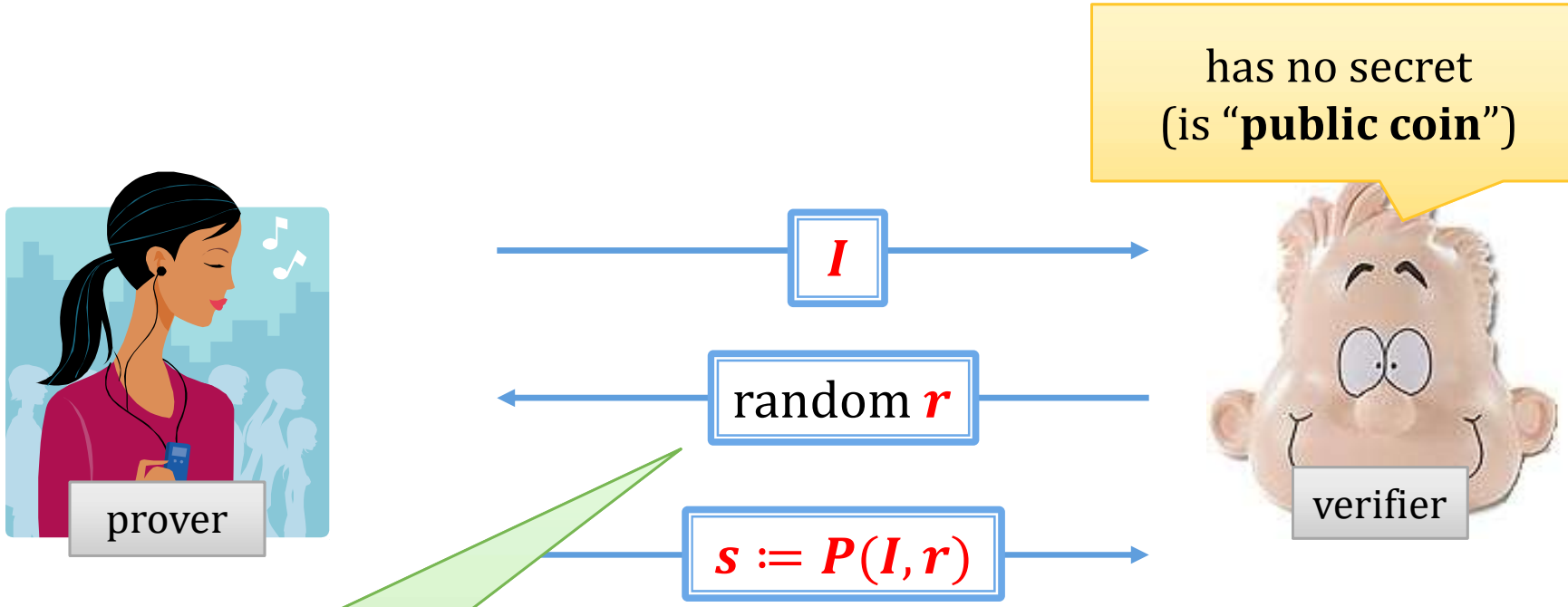
- convert standard **ZK** to **NIZK** using **Fiat-Shamir Transform** (≈ **Random Oracle Model**)
- can also be constructed just by assuming a **Common Reference String**

Exists for all languages in **NP**!

(everything efficiently provable can be proven using **NIZK**)

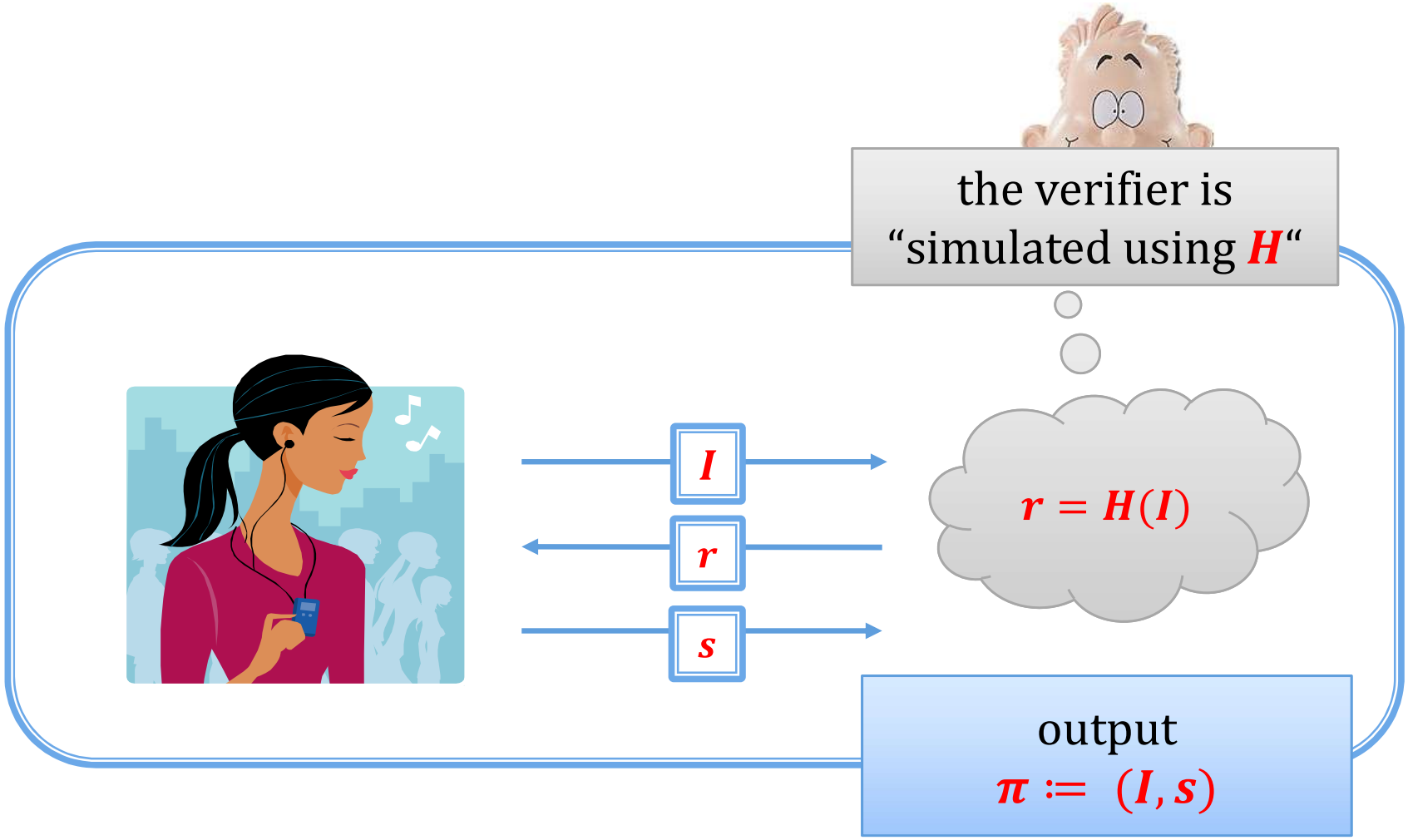
Fiat-Shamir transform: main idea

Suppose we have an **Interactive Proof** system of this form:



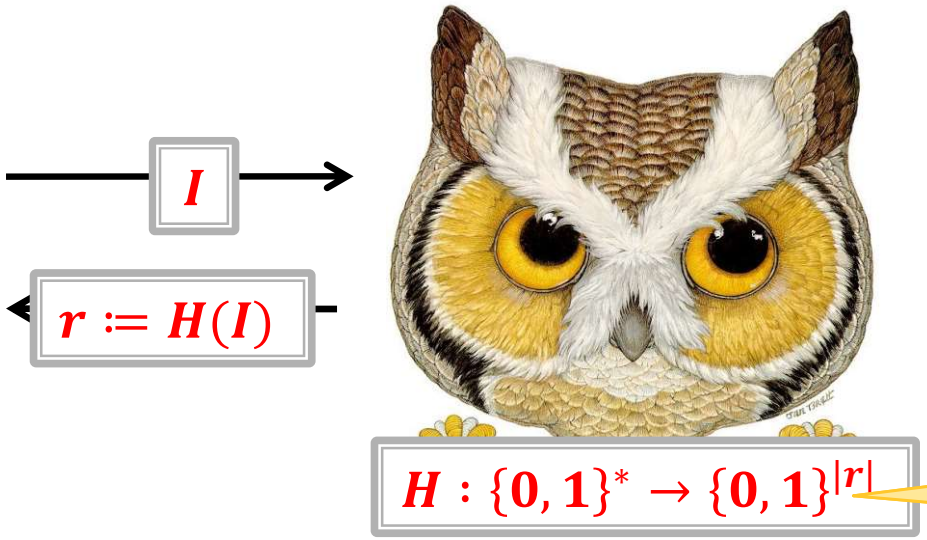
Main idea: simulate the verifier "in the head:

Let $H: \{0,1\}^* \to \{0,1\}^{|r|}$ be a **hash function**



Why does it work?

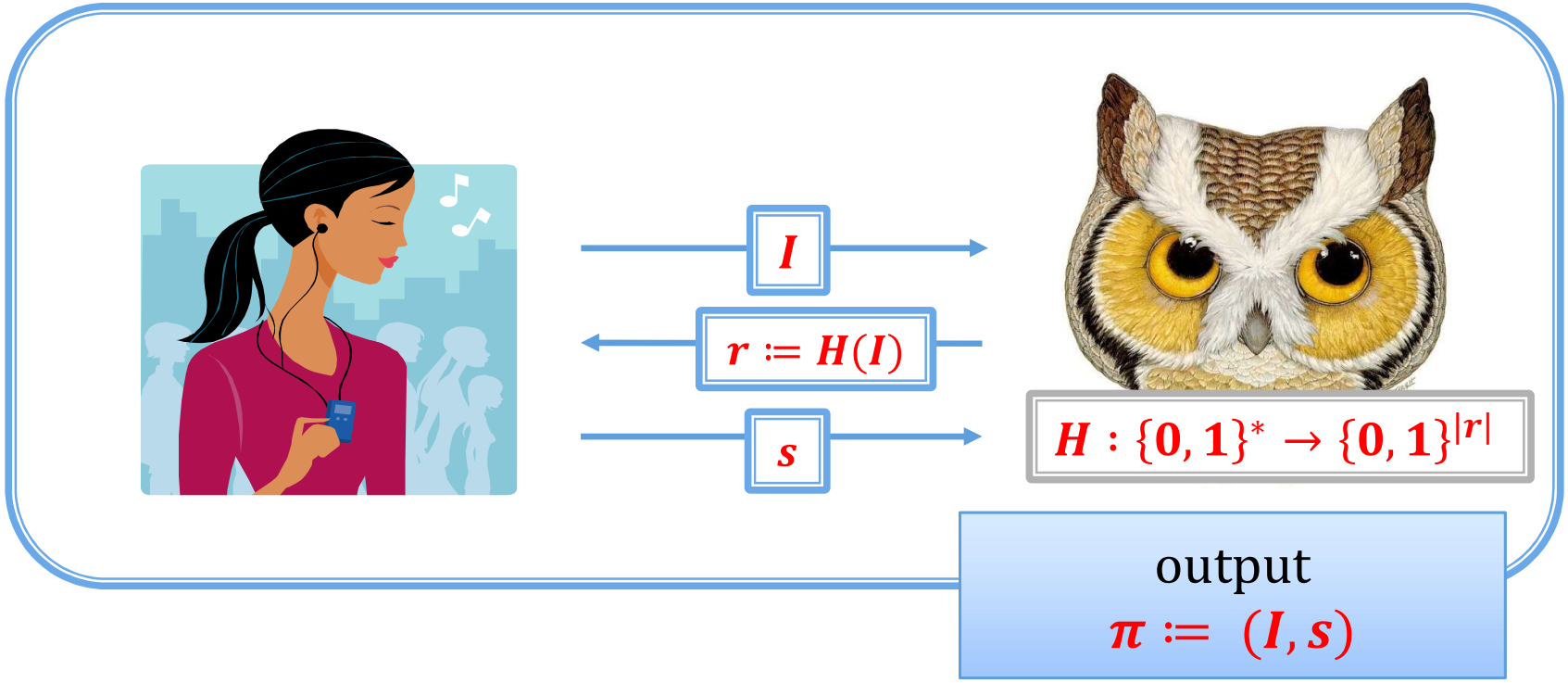
Suppose H is modelled as a Random Oracle:



a completely random function

Then it is basically the same as using the real verifier!

Pictorially



More formally

To compute **NIZK** for x:

1. use the **Prover** to sample I
2. compute $r := H(I)$
3. compute $s := P(I, r)$
4. output $\pi := (I, s)$

To **verify** if (I, s) is a **NIZK** proof for x:

1. compute $r := H(I)$
2. accept if $V(r, s) = I$

Some remarks

- it only works if ***r*** is **long** and **uniformly random**
- but: works also for the **Honest-Verifier ZK**
- can be extended to **multiple rounds**

Important: always remember to include all the variables when computing the hash.

(see: the recent "**Frozen Heart**" vulnerability

https://blog.trailofbits.com/2022/04/13/part-1-coordinated-disclosure-of-vulnerabilities-affecting-girault-bulletproofs-and-plonk/)

Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
    1. introduction
    2. signature schemes from ZK Proofs of Knowledge
5. Applications

Idea (wrong)

For example: $\mathbf{Gen}(1^n)$ run **GenG** to obtain G, g and q. Then: $x \leftarrow Z_q$ and $y := g^x$ and:
$pk := (G, g, q, y)$.
$sk := (G, g, q, x)$.

Suppose we have some relation R and an algorithm **Gen** that samples pairs

(sk, pk)

witness — statement

such that it's hard to compute sk from pk.

Design a **signature scheme** as follows:

- **Gen**: is the key generation algorithm
- to **sign** a message m the signer "proofs knowledge" of sk
- to **verify** the signature the verifier checks the proof

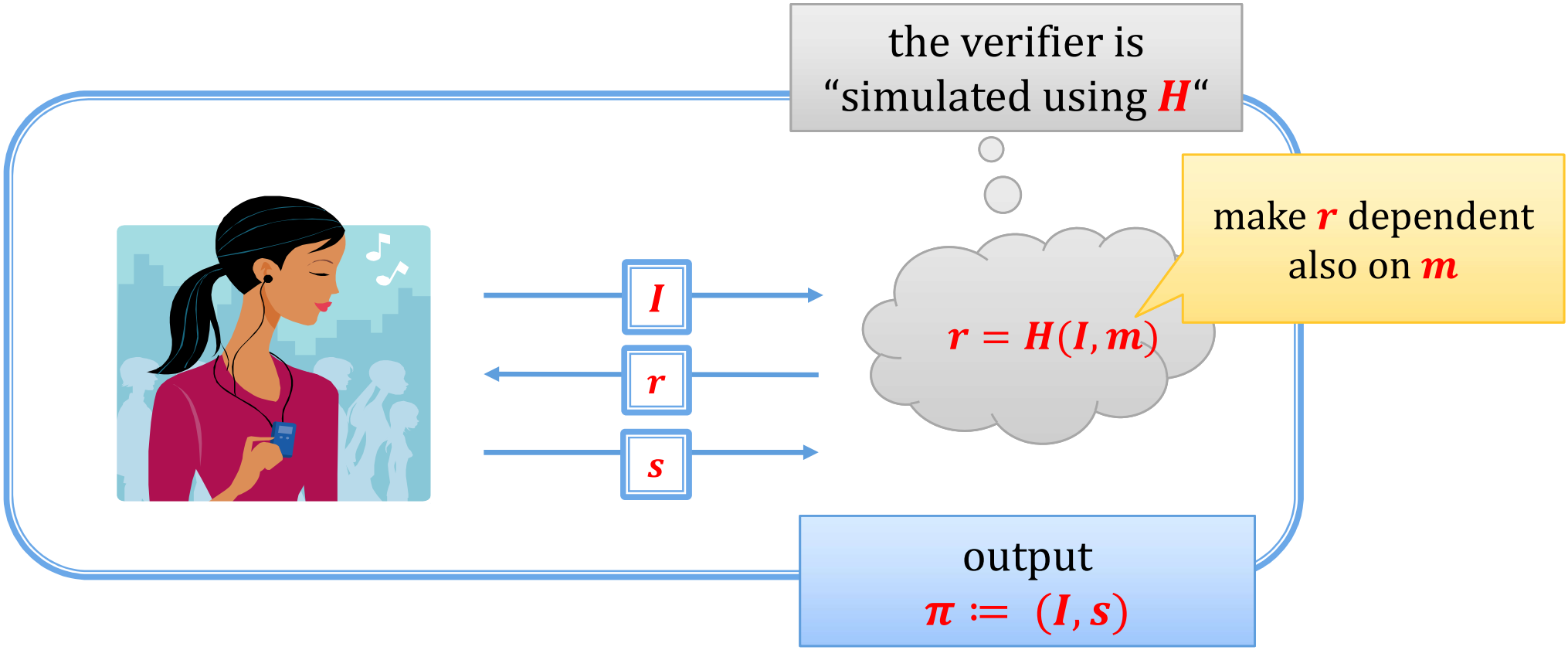
m

$\mathbf{Sign}_{sk}(m) := \pi(sk)$

makes no sense: the signature needs to be "bound" to m

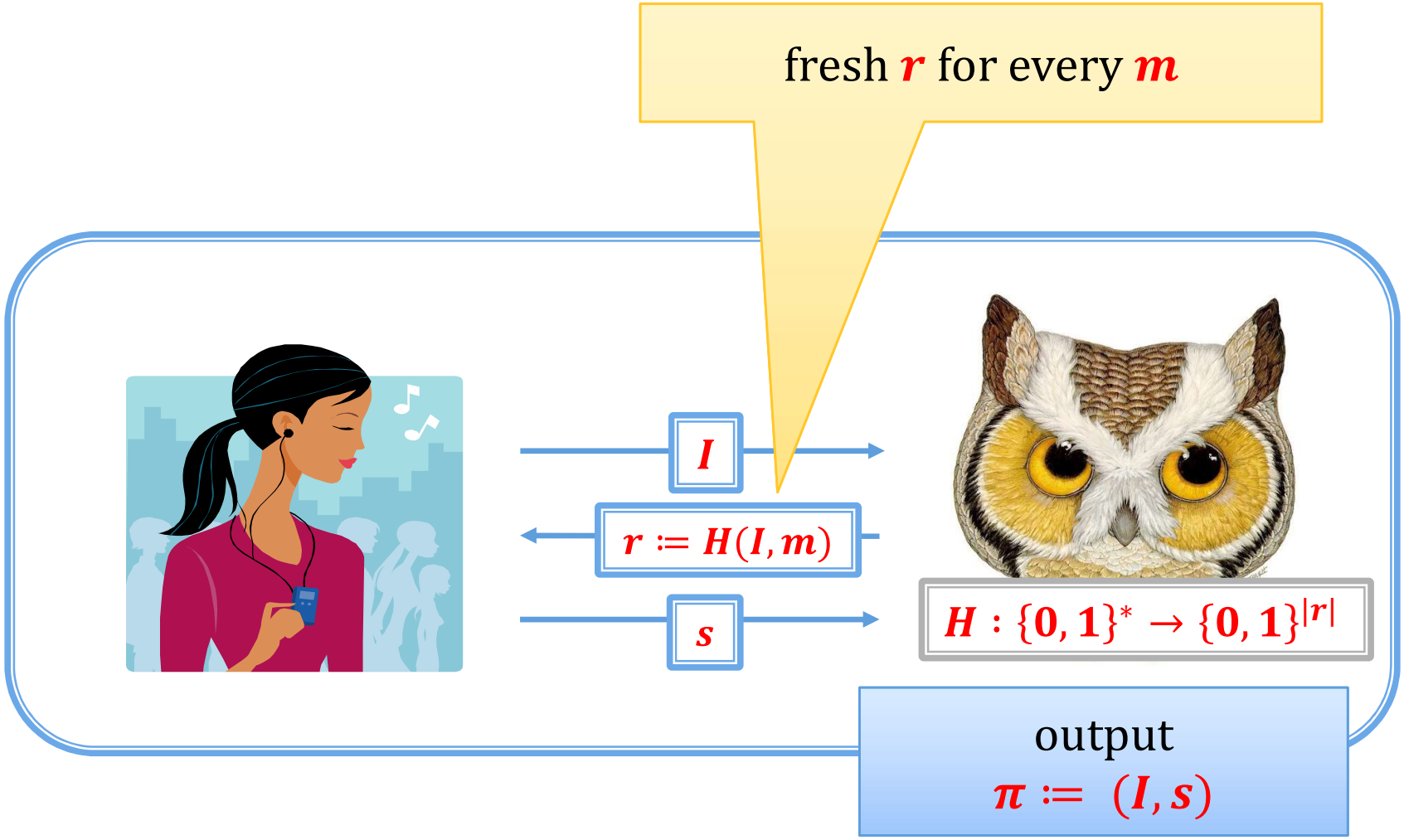
Better idea:

Suppose **NIZK** is constructed as before:



Intuition: the prover is now proving the knowledge of the witness in a way that depends on m

Pictorially



How does it look now

equivalently:

$\mathbf{Sign}_{sk}(m) :=$

1. use the **Prover** to sample I
2. compute $r := H(I, m)$
3. compute $s := P(I, r)$
4. output (I, s)

$\mathbf{Vrfy}_{pk}(m, (I, s)) :=$

1. compute $r := H(I)$
2. accept if $V(r, s) = I$

$\mathbf{Sign}_{sk}(m) :=$

1. use the **Prover** to sample I
2. compute $r := H(I, m)$
3. compute $s := P(I, r)$
4. output (r, s)

$\mathbf{Vrfy}_{pk}(m, (r, s)) :=$

1. compute $I := V(r, s)$
   accept if $r = H(I, m)$

equivalently: accept if $r = H(V(r, s), m)$

Remember the Schnorr's signatures?

They are simply:

Schnorr's ZK proof of knowledge converted to a signature scheme using the Fiat-Shamir transform!

Let's look at them again…

Schnorr's signature scheme

$\mathbf{Gen}(1^n)$ run **GenG** to obtain G, g and q (assume q is prime). Then, choose $x \leftarrow Z_q$ and computes $y := g^x$.

- The public key pk is (G, g, q, y).
- The private key sk is (G, g, q, x).

$\mathbf{Sign}(sk, m)$:

1. choose uniform $k \leftarrow Z_q$ and let $I := g^k$
2. compute $r := H(I, m)$
3. compute $s := rx + k \bmod q$
4. output (r, s)

$\mathbf{Vrfy}(pk, m, (r, s))$:

output **yes** if $r = H(g^s \cdot y^{-r}, m)$.

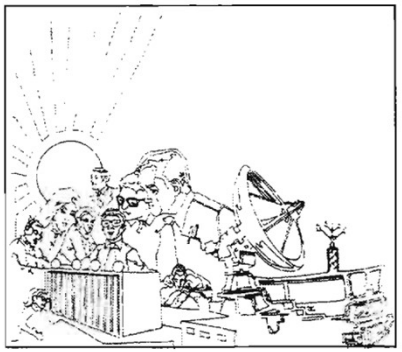
Plan

1. Interactive Proofs
2. Zero-Knowledge Proofs
3. Zero-Knowledge Proofs of Knowledge
4. Non-Interactive Zero-Knowledge
5. Applications

Zero-Knowledge

Initially considered by some practitioners a part of **philosophy**, not **computer science**.

For example, look at these declassified **NSA documents**.



NATIONAL SECURITY AGENCY

CRYPTOLOG

DOCID: 4009689
CRYPTOLOG
March 1994

~~CONFIDENTIAL~~

P.L. 86-36

EUROCRYPT '92

(U) Those of you who know my prejudice against the "zero-knowledge" wing of the philosophical camp will be surprised to hear that I enjoyed the three talks of the session better than any of that ilk that I had previously endured. The reason is simple: I took along some interesting reading material and ignored the speakers.

https://documents.epfl.ch/users/l/le/lenstra/public/papers/cryptolog_126.pdf

Initial vision for applications of ZK

Example:

Alice has a document (signed by some public authority) saying:

"Alice was born on **DD-MM-YYYY**".

She can now prove in zero-knowledge that

she is at least **18** years old (without revealing her exact age)

Not very convincing for practitioners ("we can do it more efficiently with trusted hardware")

Another natural application

Proving that your program “behaves well” without revealing the details.

(no need for external auditor)

Also **remained mostly theoretical**…

Applications within cryptography

A **theoretical application**:

- designing **CCA-secure encryption** (use NIZK to prove the "knowledge of a ciphertext")

A more **practical** one:

- converting **passively-secure to actively-secure MPCs** (see next lecture)

Everything changed with this

A very popular tool in this space: ZK-SNARKS

for some applications ZK is not needed

short proofs

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

secure only against **computationally bounded provers**

verification time measured in **milliseconds**, proof fits on a **QR code**

Eli Ben-Sasson, et al. **SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge**

Survey: https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

One problem with initial SNARKs: required "**trusted setup**" ("**toxic waste**")

The first spectacular application

ZCASH

"**Privacy-protecting digital currency**"

"**Trusted setup**" was done using **MPCs** (see next lecture) in a "**Parameter Generation Ceremony**"

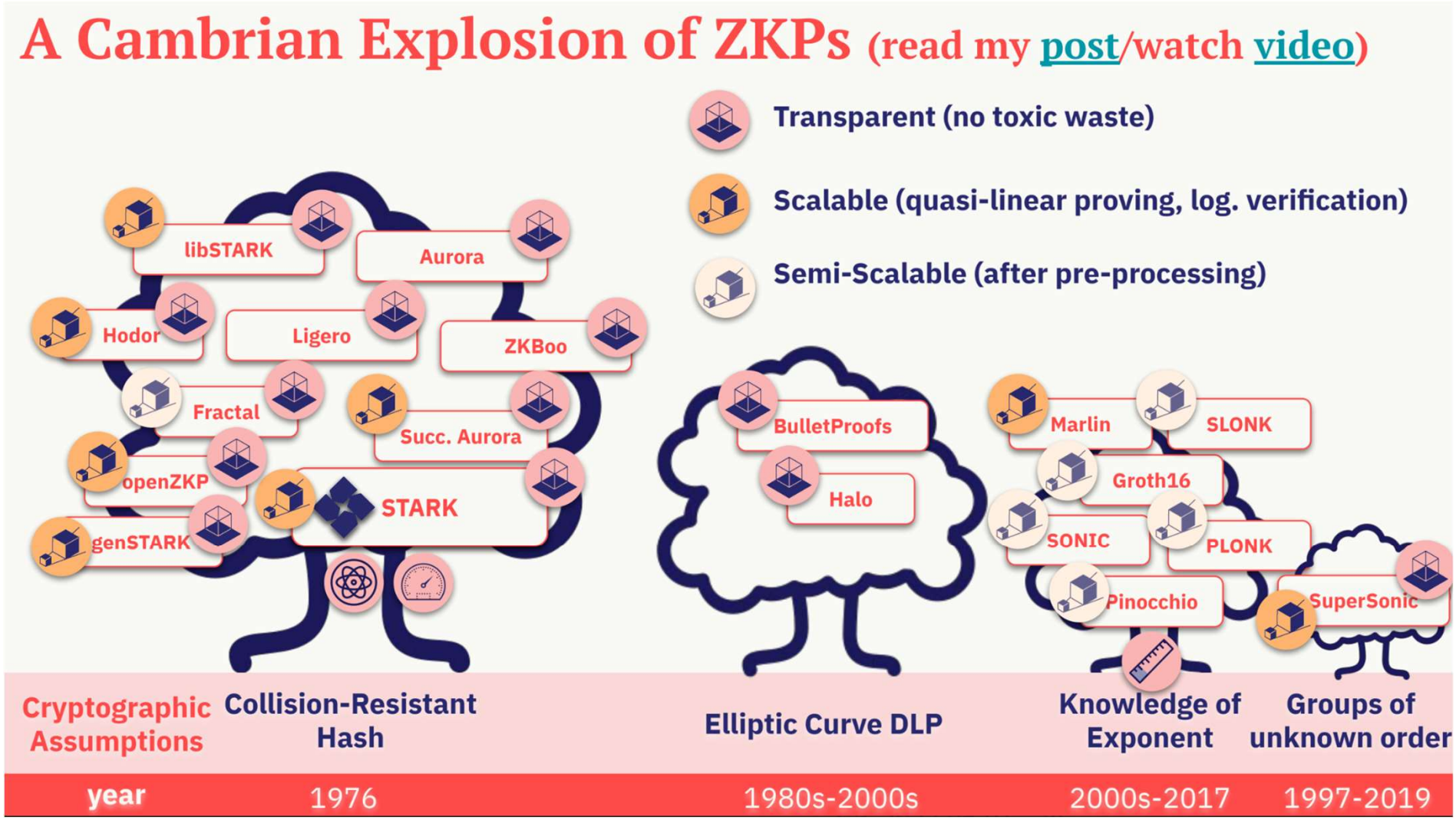
Several blockchain applications

Blockchain 2.0 – a large **computing platform** maintained in a **distributed way** via a "**consensus algorithm**".

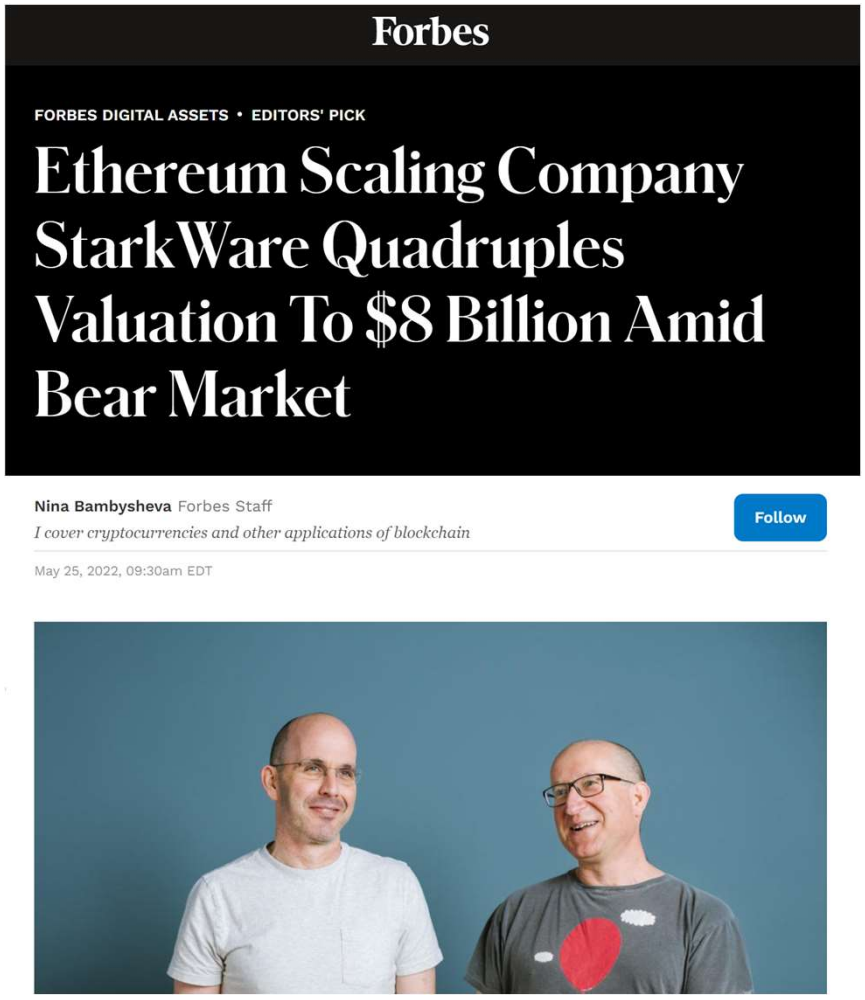
executing programs is **slow** and **expensive**

SNARKs allow to execute code privately and succinctly prove correctness!

A slide from a talk by **Eli Ben-Sasson** (Starkware)



Philosophy?



Forbes

FORBES DIGITAL ASSETS • EDITORS' PICK

Ethereum Scaling Company StarkWare Quadruples Valuation To $8 Billion Amid Bear Market

Nina Bambysheva Forbes Staff

I cover cryptocurrencies and other applications of blockchain

Follow

May 25, 2022, 09:30am EDT

There is nothing more practical than a good theory!