Cryptographic Hardness of PPAD via Non-Interactive Arguments


Pavel Hubáček


Chethan Kamath


Krzysztof Pietrzak


Alon Rosen


Guy Rothblum


Nir Bitansky


Justin Holmgren


Alex Lombardi

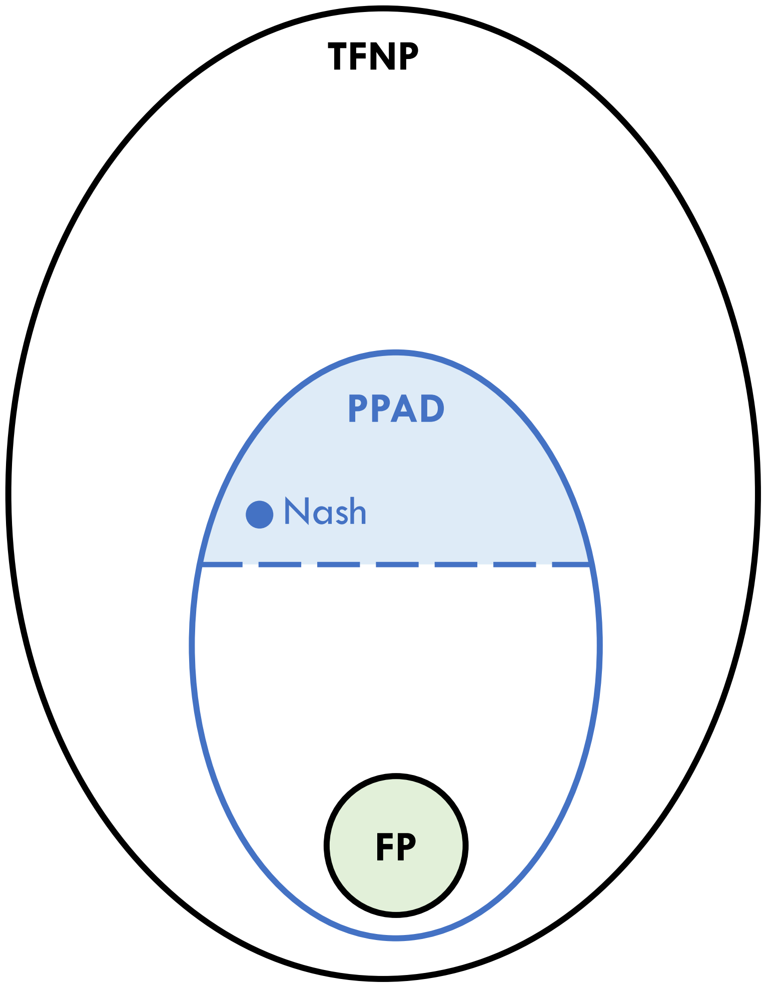

Omer Paneth


Ron Rothblum

Arka Rai Choudhuri

Polynomial-Parity Argument on Digraphs (PPAD)

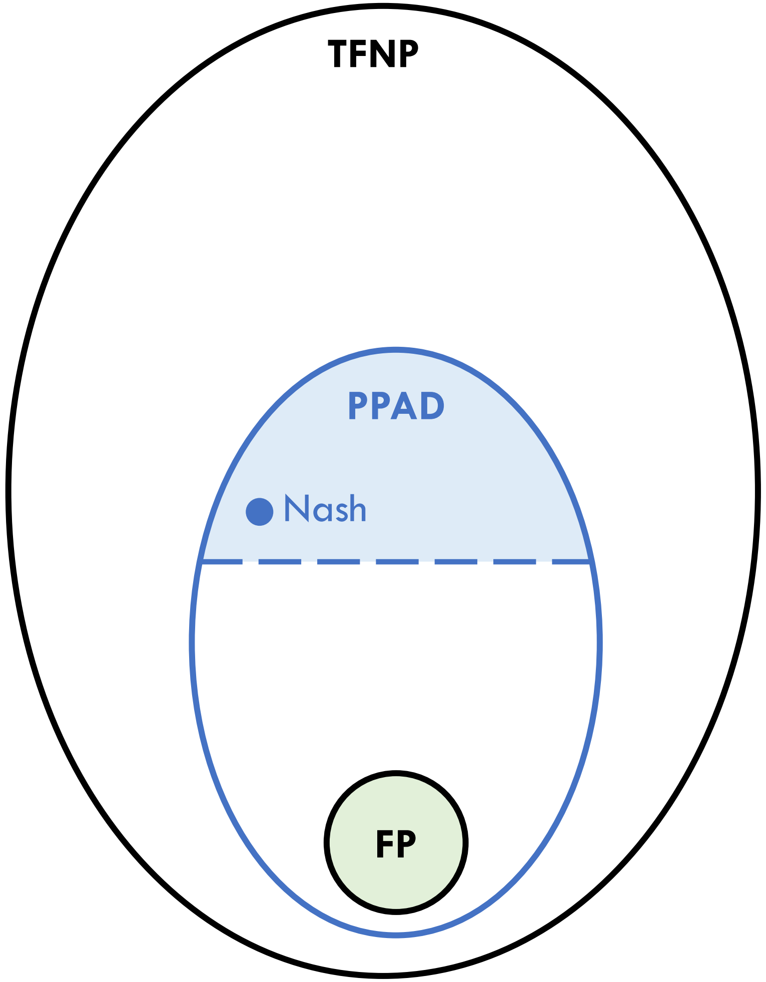
Class of Total Search Problems



Polynomial-Parity Argument on Digraphs (PPAD)

Class of Total Search Problems

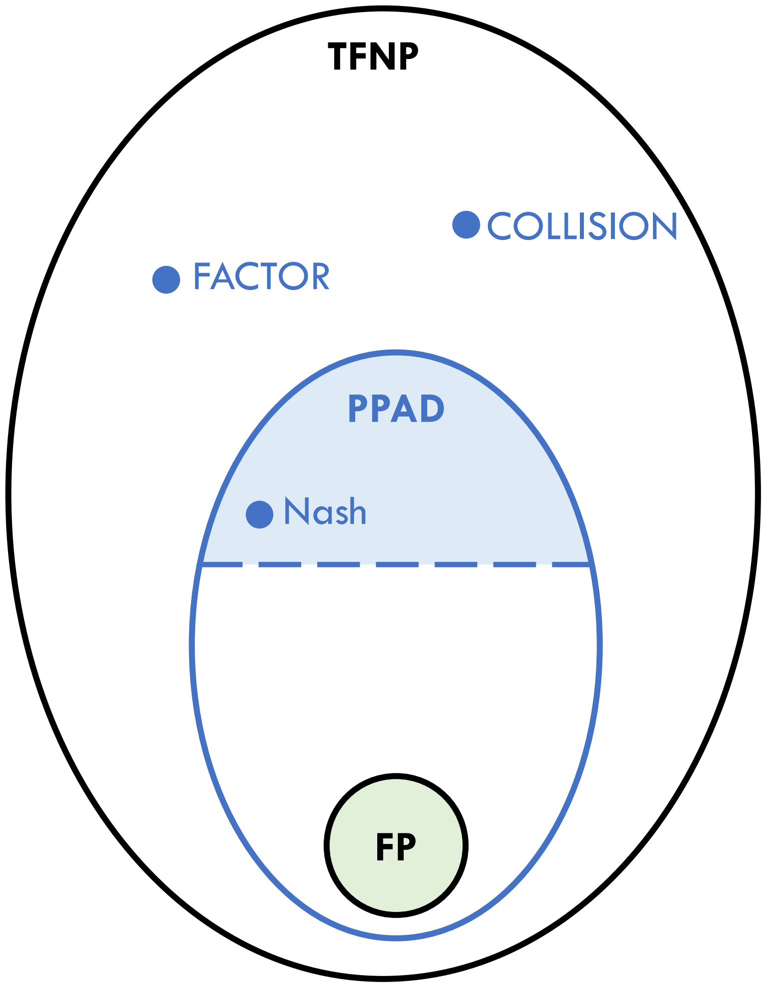
Can we use Cryptography to show PPAD Hardness?



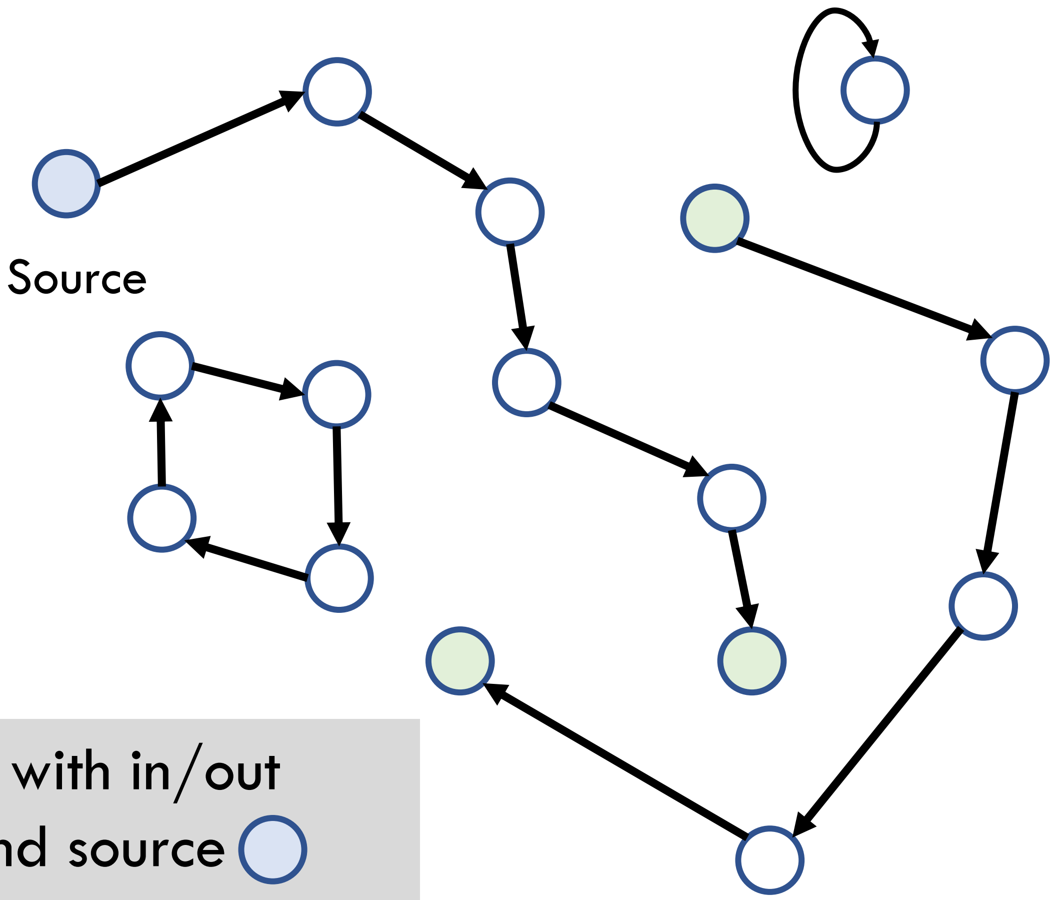
Polynomial-Parity Argument on Digraphs (PPAD)

Class of Total Search Problems

Can we use Cryptography to show PPAD Hardness?

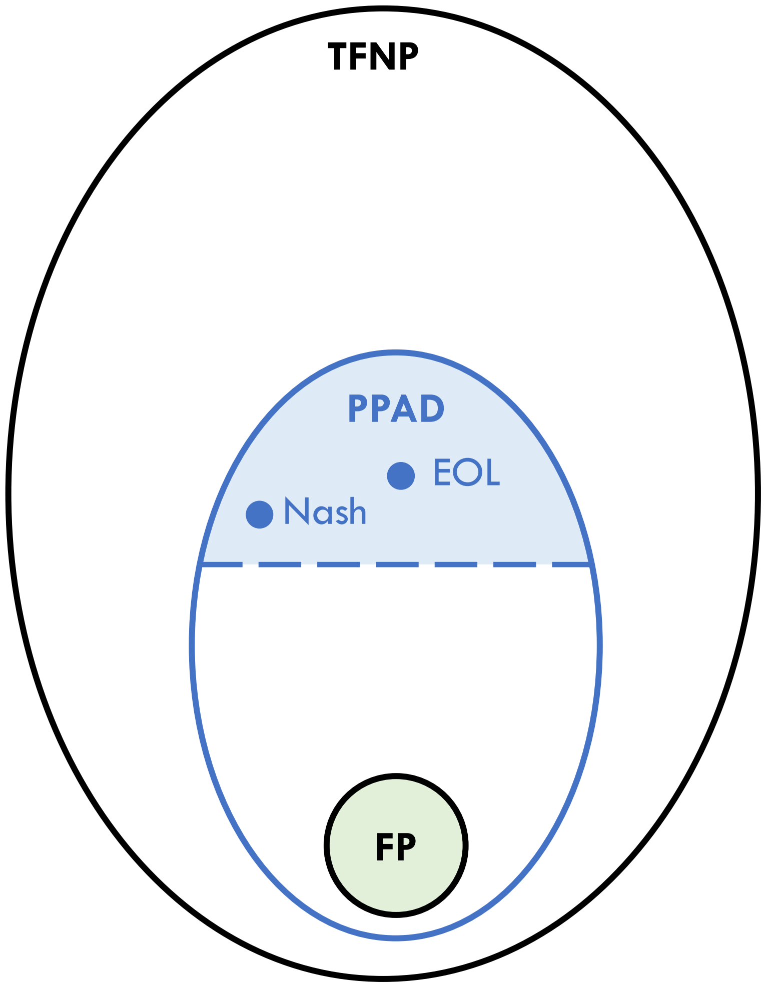


End of Line (EOL)

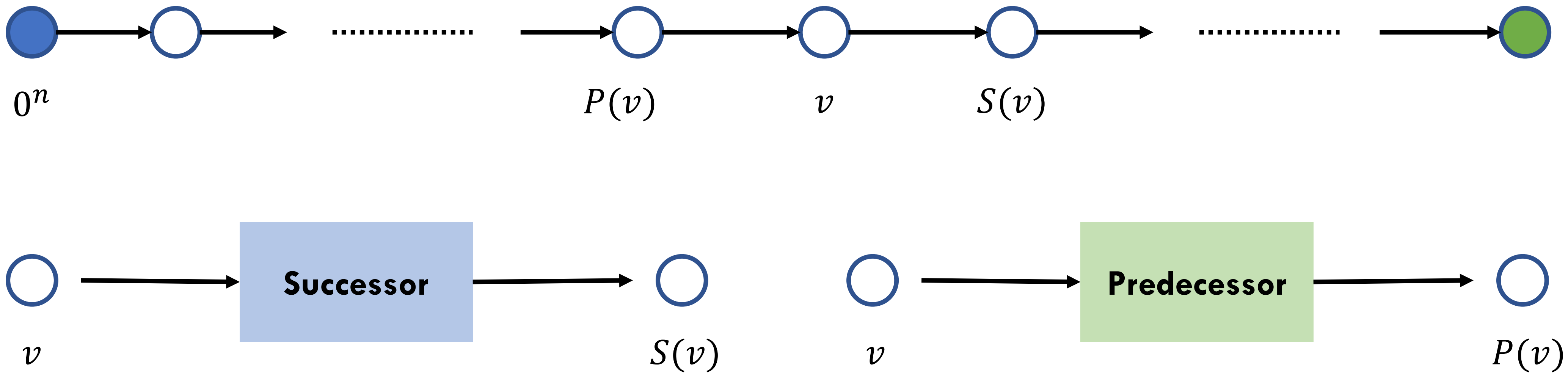


Input: A graph with in/out degree ≤ 1 and source

Output: Another source/sink



End of Line (EOL)

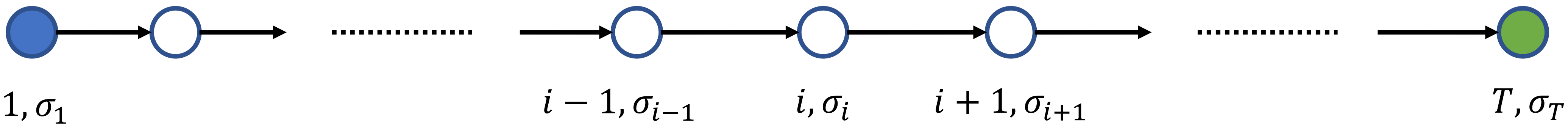


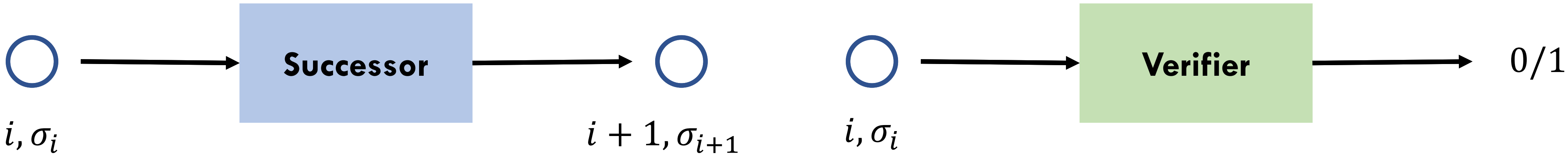
Goal: Find v such that

$$P(S(v)) \neq v \text{ or } S(P(v)) \neq v \neq 0^n$$

Sink of Verifiable Line (SVL)

[Abbott-Kane-Valiant'04,Bitansky-Paneth Rosen'15]





Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

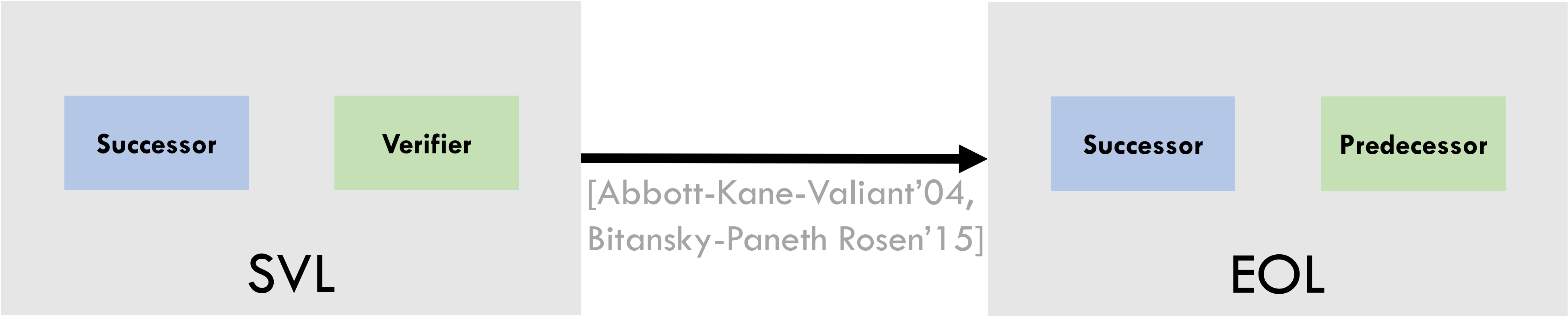
$$\text{Verifier}(T, \sigma_T) = 1$$

Promise:

$$\text{Verifier}(i, \sigma_i) = 1 \iff \text{Successor}^{i-1}(1, \sigma_1)$$

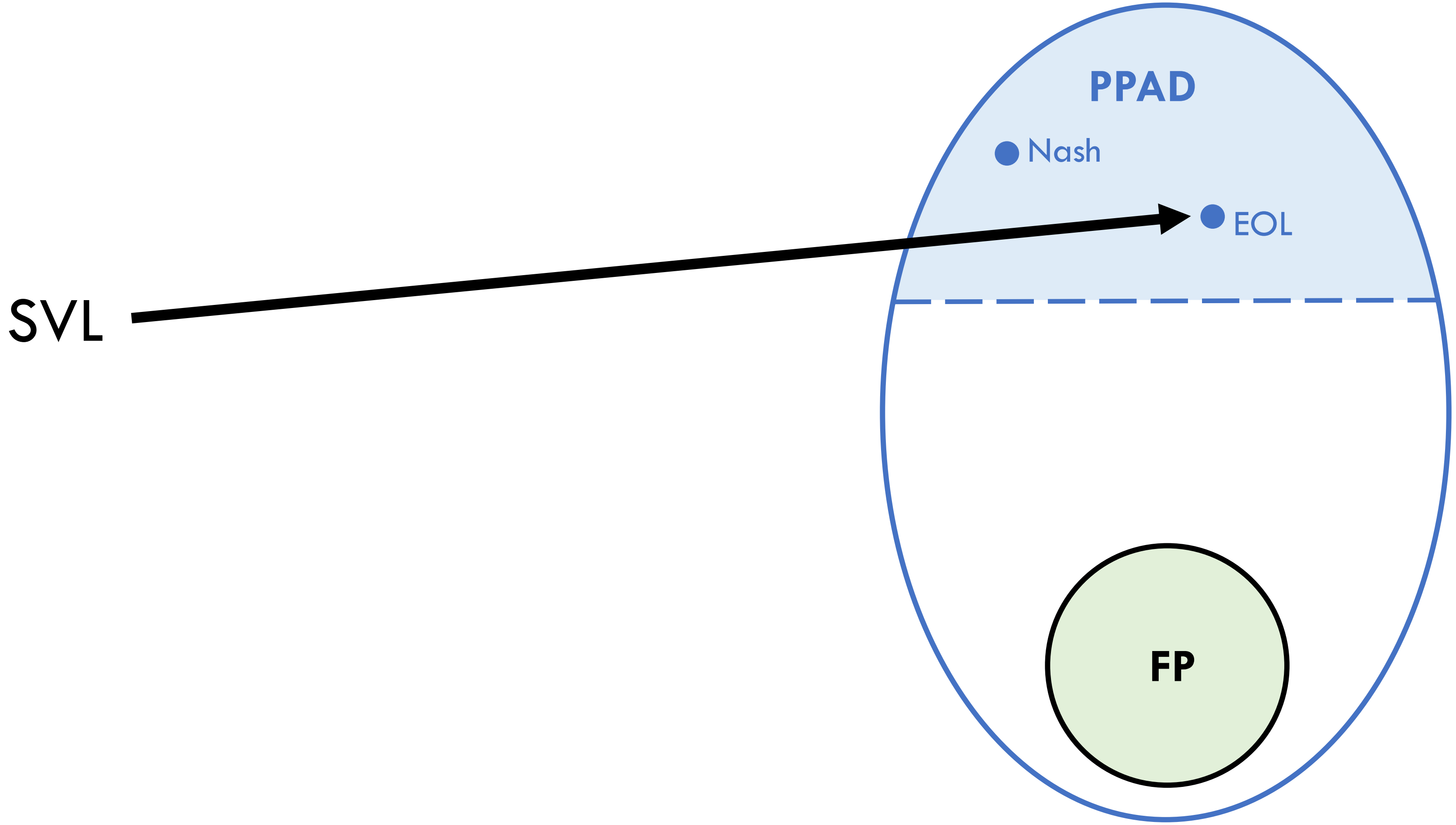
SVL not in TFNP

SVL Reduces to EOL

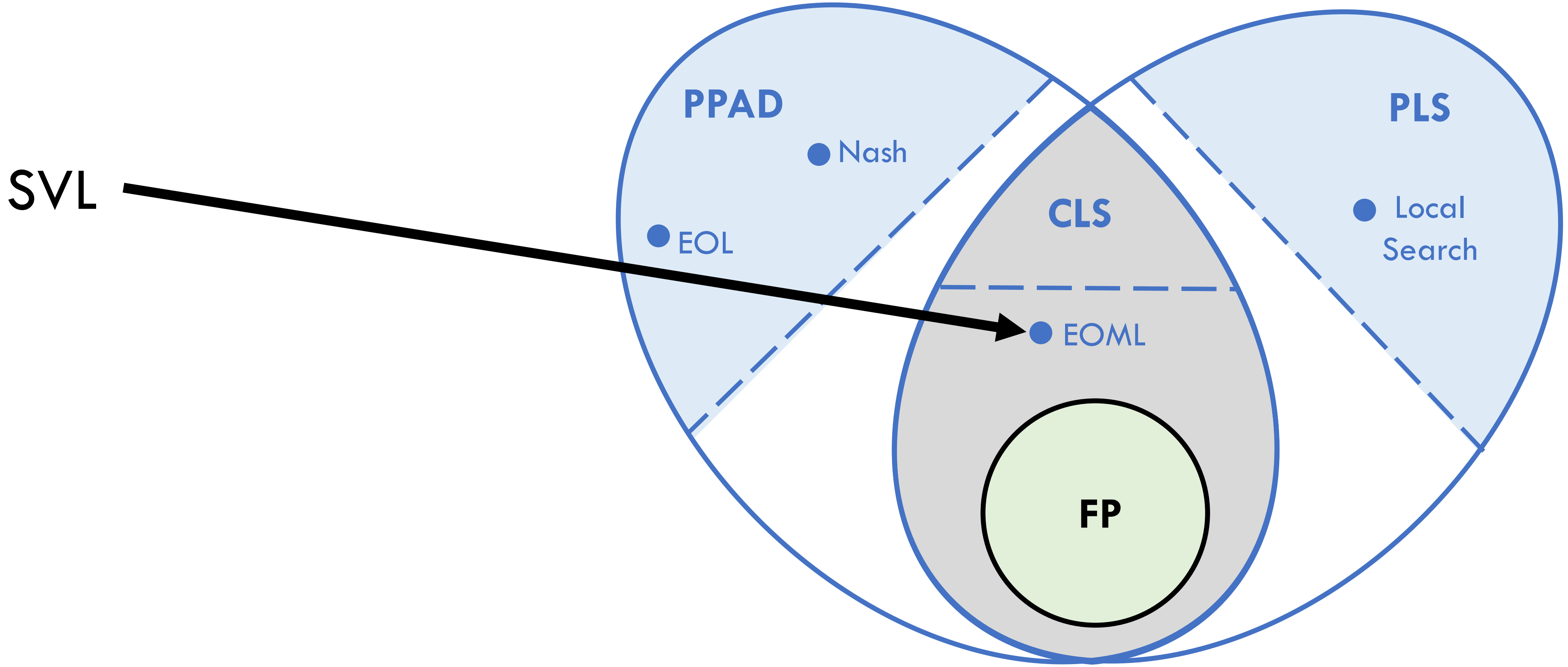


If path is verifiable, then Predecessor is for free. Use [Bennett'89] ideas of reversible computation via pebbling.

SVL Reduces to EOL



SVL Reduces to EOML

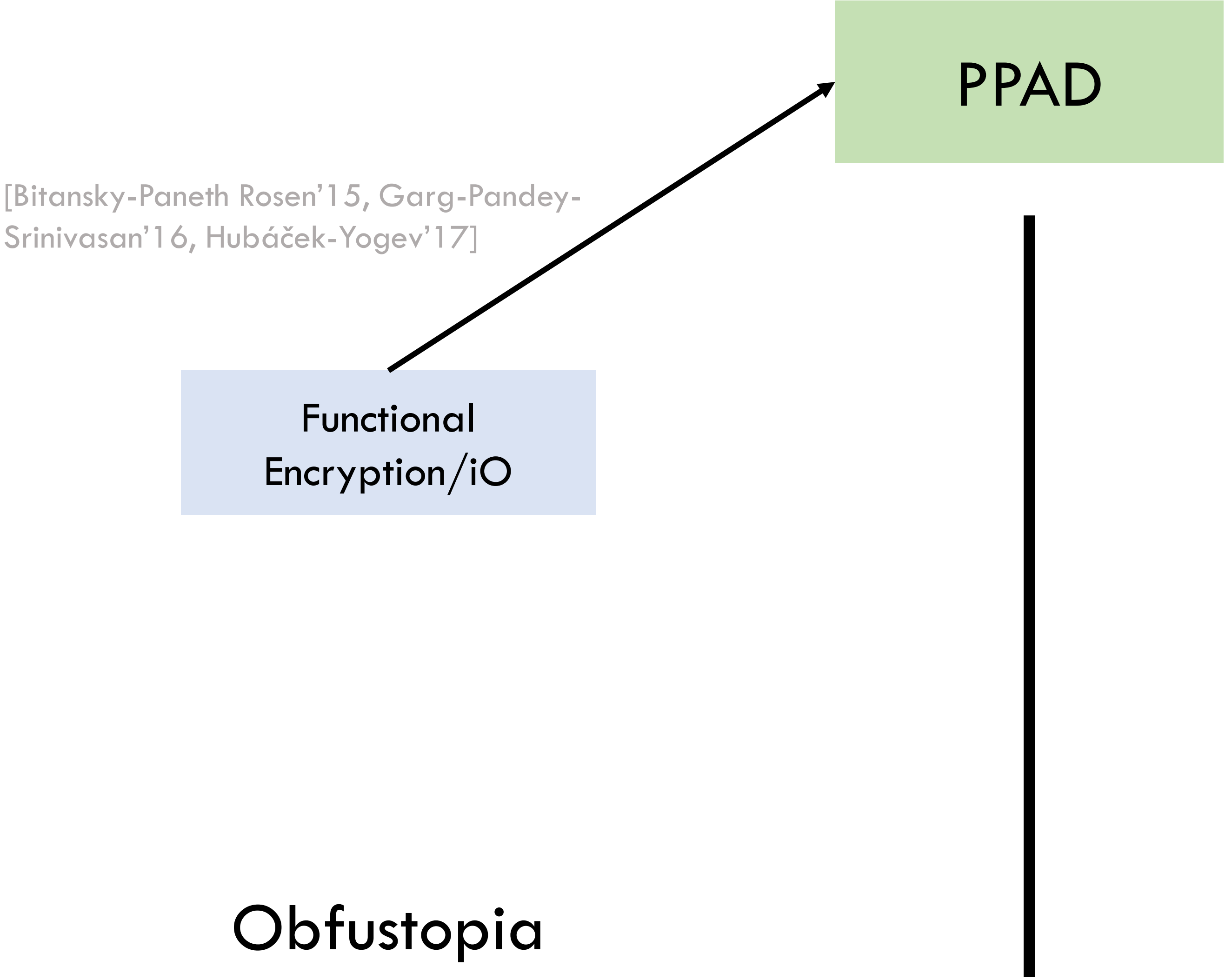


PPAD Hardness from Standard Cryptographic Assumptions

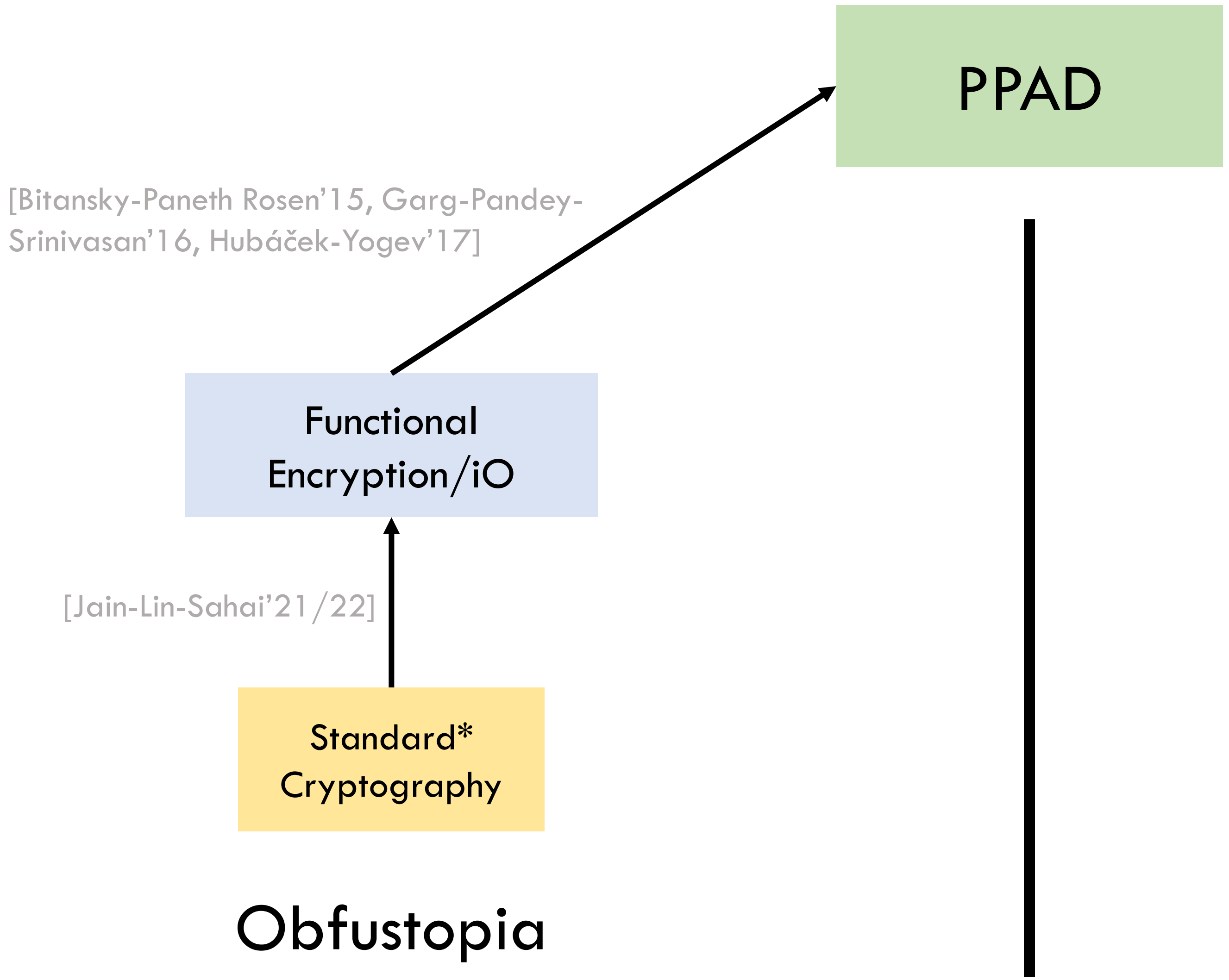
PPAD

Obfustopia

PPAD Hardness from Standard Cryptographic Assumptions



PPAD Hardness from Standard Cryptographic Assumptions



PPAD Hardness from Standard Cryptographic Assumptions

PPAD

[Bitansky-Paneth Rosen'15, Garg-Pandey-Srinivasan'16, Hubáček-Yogev'17]

Functional

$g \in \mathbb{G}$, $T \in \mathbb{N}$ → Solve → g^{2^T}

$\mathbb{G}$ is a group of unknown order

[Jain-Lin-Saha

IS – Iterated Squaring

Obfustopia

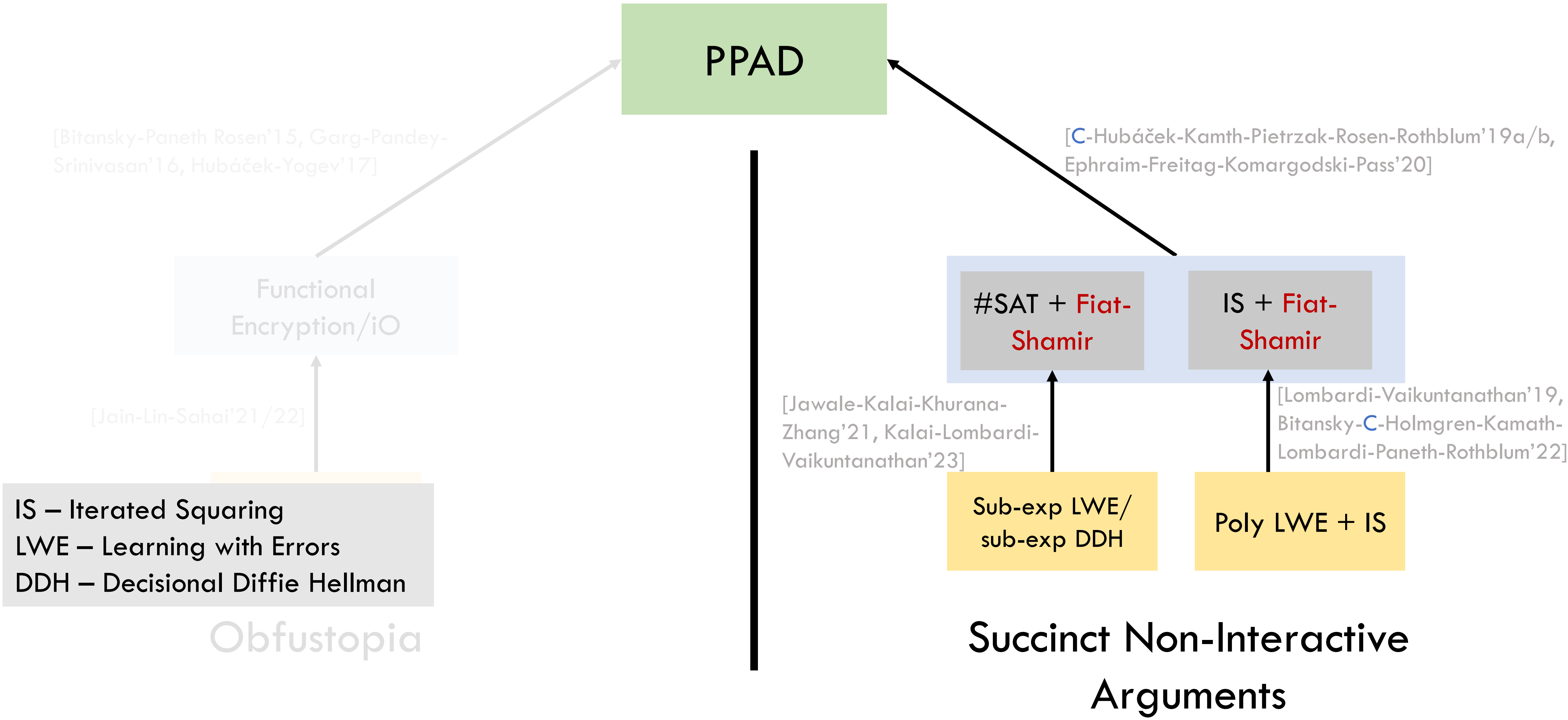
[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum'19a/b, Ephraim-Freitag-Komargodski-Pass'20]

#SAT + Fiat-Shamir

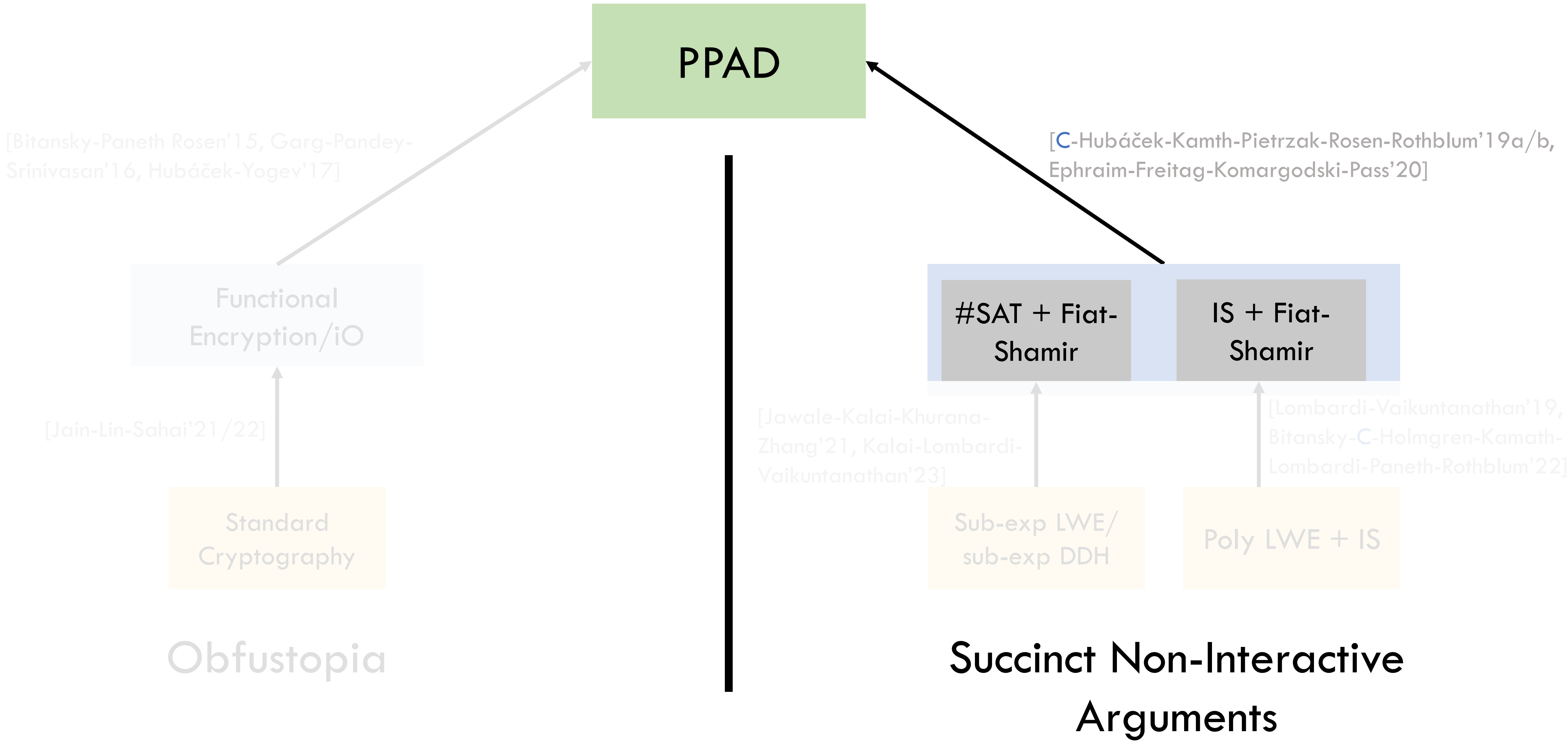
IS + Fiat-Shamir

Succinct Non-Interactive Arguments

PPAD Hardness from Standard Cryptographic Assumptions



PPAD Hardness from Standard Cryptographic Assumptions

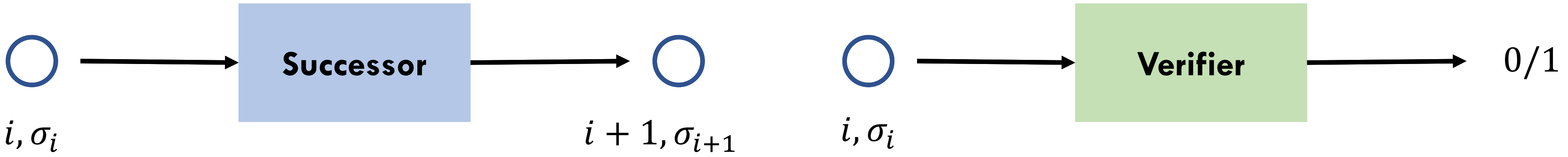


Obfuscation Approach to EOL Hardness

[Bitansky-Paneth Rosen'15, Garg-Pandey-Srinivasan'16, Hubáček-Yogev'17]

1. Generate labels σ_i to be pseudorandom (PRF).
2. Obfuscate Successor and Verifier to hide PRF key.

Intuition: Must make "oracle" calls to traverse the graph.



Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

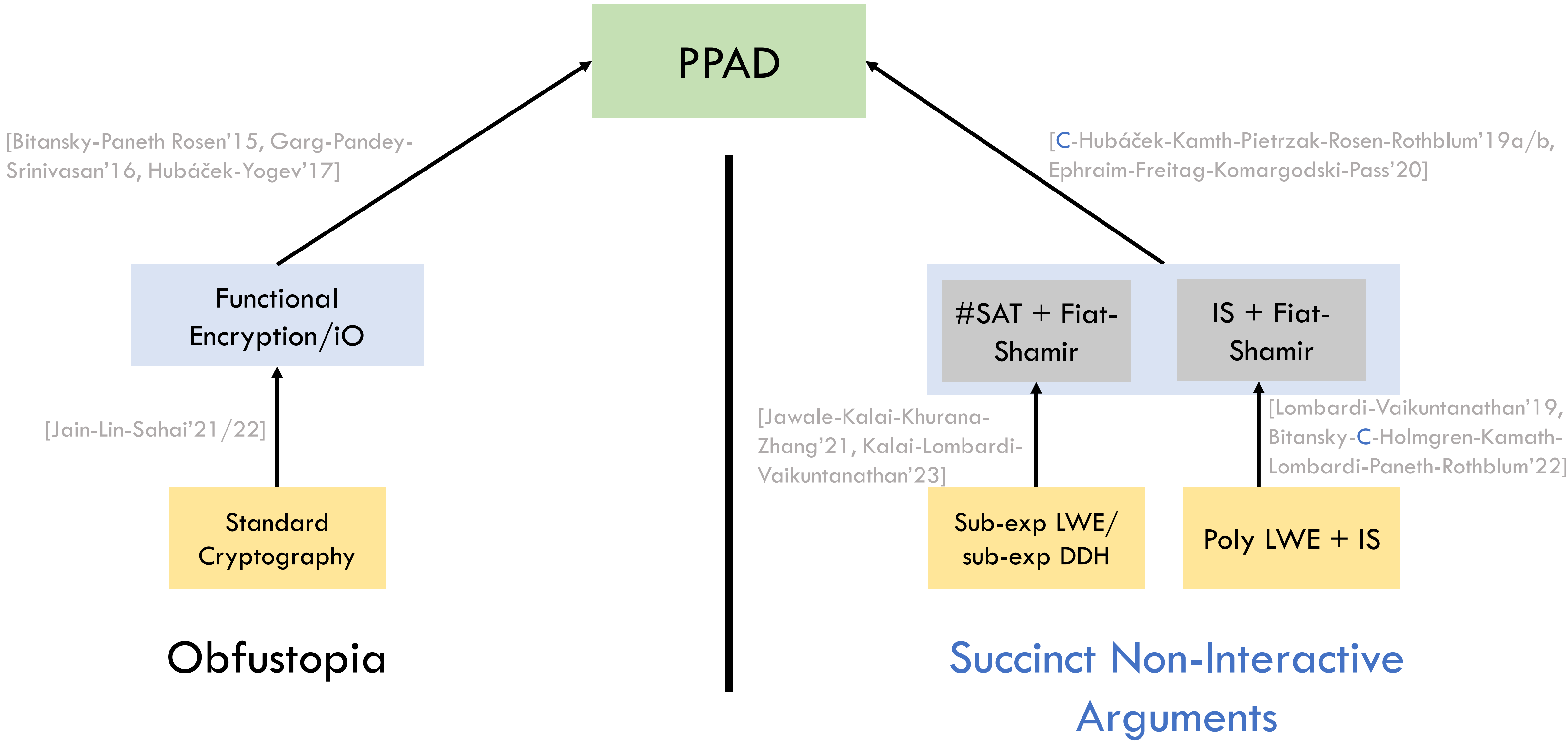
$$\text{Verifier}(T, \sigma_T) = 1$$

Promise:

$$\text{Verifier}(i, \sigma_i) = 1 \iff \text{Successor}^{i-1}(1, \sigma_1)$$

SVL not in TFNP

PPAD Hardness from Standard Cryptographic Assumptions



Succinct Non-Interactive Arguments (SNARGs)

$\mathcal{M}$, x

$\mathcal{M}$, x

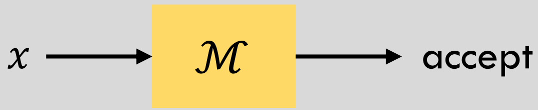
$x \longrightarrow \mathcal{M} \longrightarrow$ accept

within T steps

Succinct Non-Interactive Arguments (SNARGs)

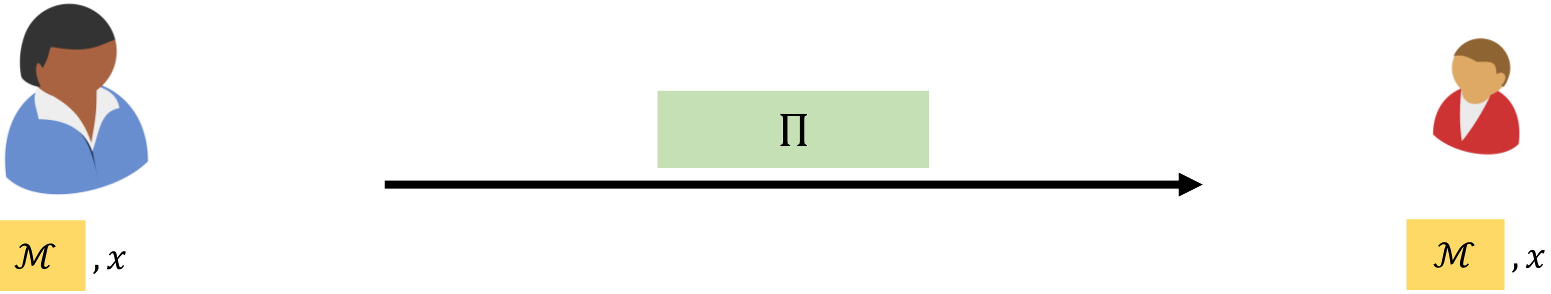
$\mathcal{M}$, x

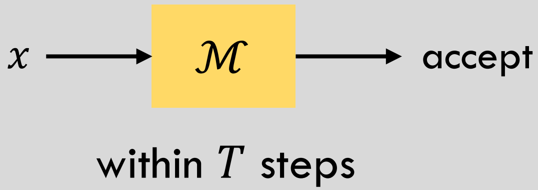
$\mathcal{M}$, x



 wants to delegate computation to 

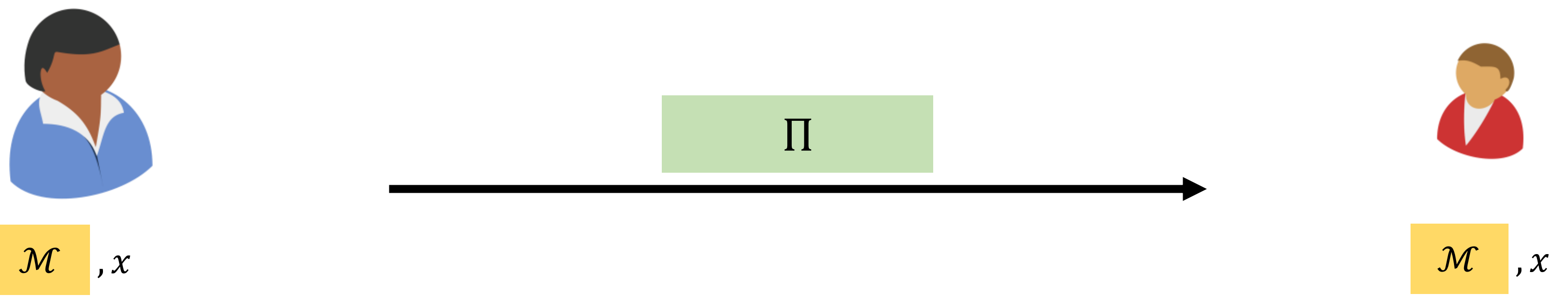
Succinct Non-Interactive Arguments (SNARGs)

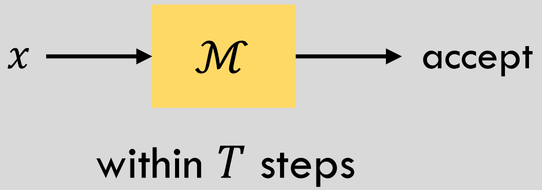




Succinct Non-Interactive Arguments (SNARGs)

Common Reference String (CRS)





Succinct Non-Interactive Arguments (SNARGs)

Common Reference String (CRS)

Π

$\mathcal{M}$, x

$\mathcal{M}$, x

Π is publicly verifiable

$x \longrightarrow \mathcal{M} \longrightarrow$ accept

within T steps

Succinct Non-Interactive Arguments (SNARGs)

Common Reference String (CRS)

$\mathcal{M}$, x

$\leftarrow$ polylog(T) $\rightarrow$

Π

$\mathcal{M}$, x

Verifier running time: polylog(T)

Π is publicly verifiable

$x \rightarrow \mathcal{M} \rightarrow$ accept

within T steps

Succinct Non-Interactive Arguments (SNARGs)

Common Reference String (CRS)

$\mathcal{M}$, x

$\leftarrow$ polylog(T) $\rightarrow$

Π

$\mathcal{M}$, x

Verifier running time: polylog(T)

Π is publicly verifiable

$x \longrightarrow \mathcal{M} \longrightarrow$ accept

within T steps

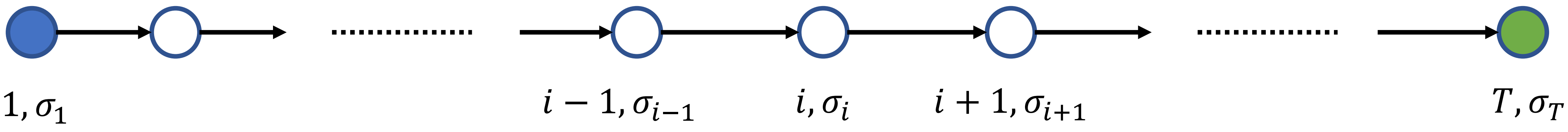
No PPT  can produce accepting Π if

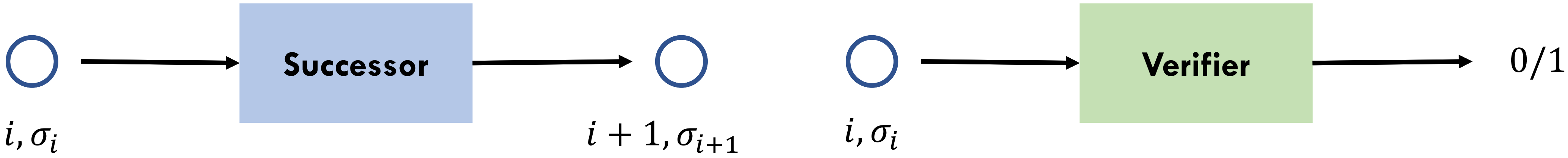
$x \longrightarrow \mathcal{M} \not\longrightarrow$ accept

within T steps

Sink of Verifiable Line (SVL)

[Abbott-Kane-Valiant'04,Bitansky-Paneth Rosen'15]





Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

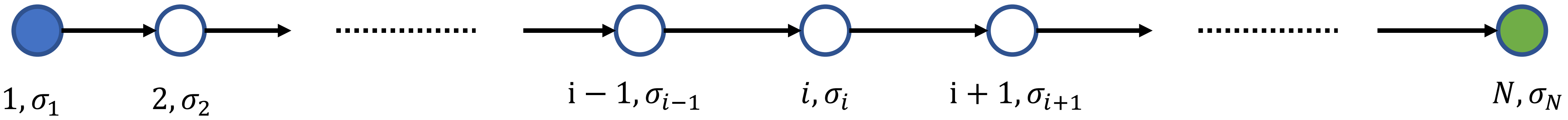
$$\text{Verifier}(T, \sigma_T) = 1$$

Promise:

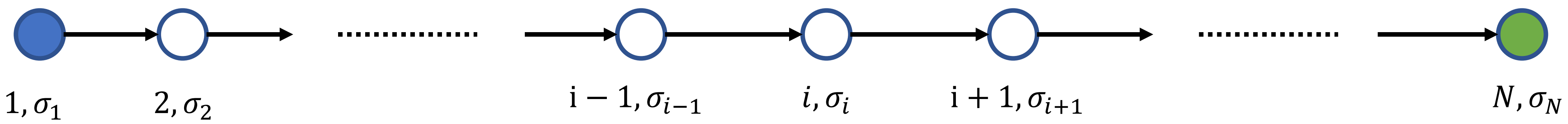
$$\text{Verifier}(i, \sigma_i) = 1 \iff \text{Successor}^{i-1}(1, \sigma_1)$$

SVL **not** in TFNP

Basic Idea: Long Computation + SNARGs

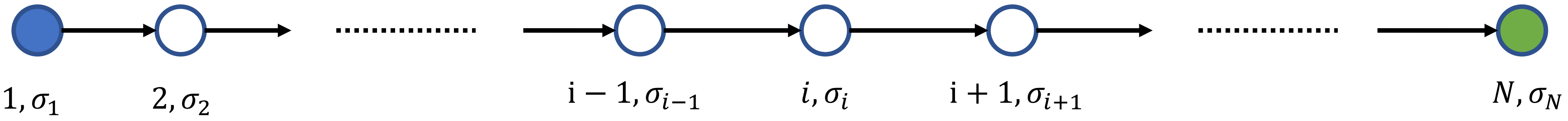


Basic Idea: Long Computation + SNARGs

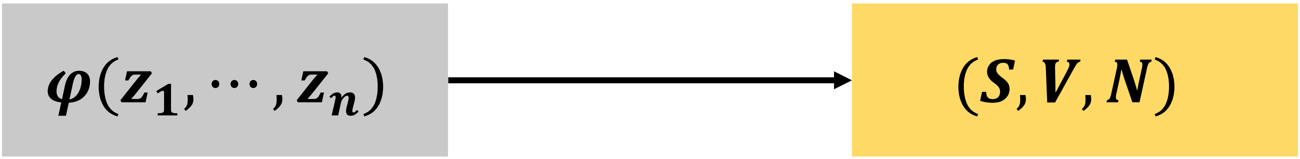


Reduce to SVL from #SAT

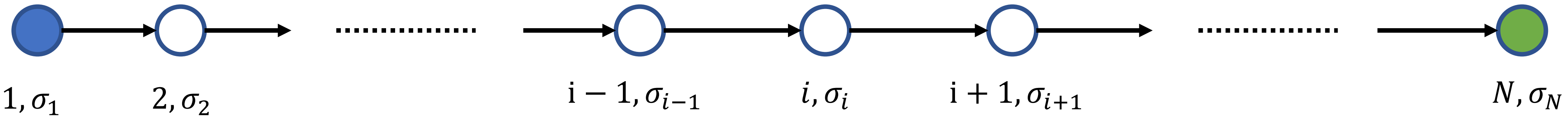
Basic Idea: Long Computation + SNARGs



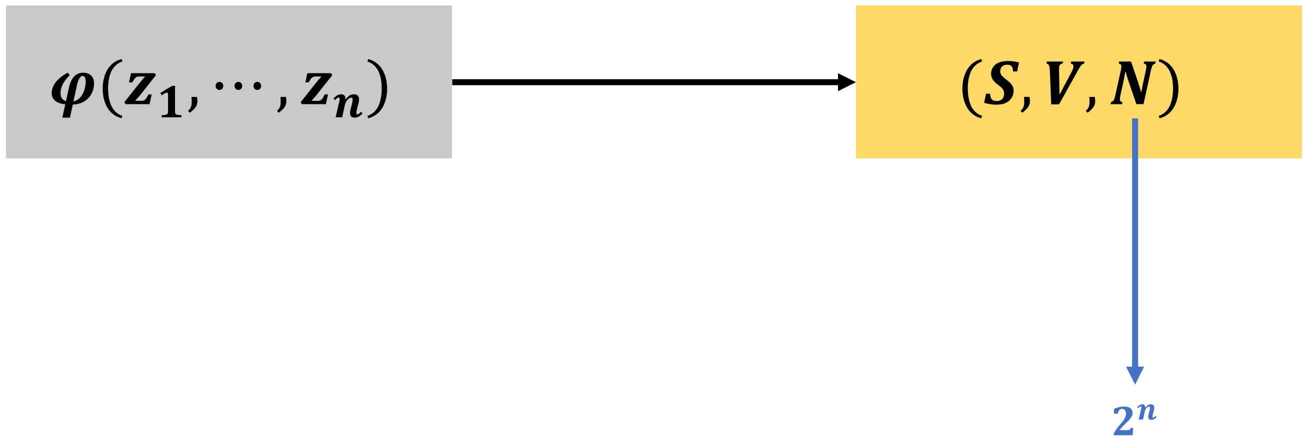
Reduce to SVL from #SAT



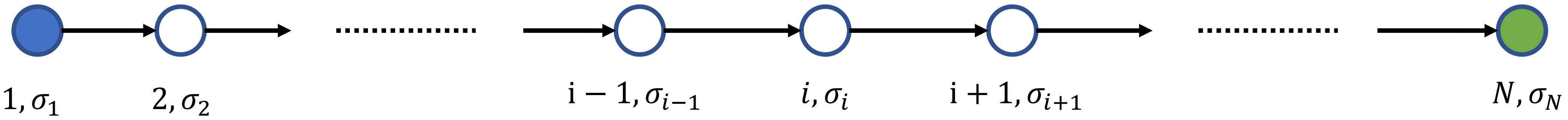
Basic Idea: Long Computation + SNARGs



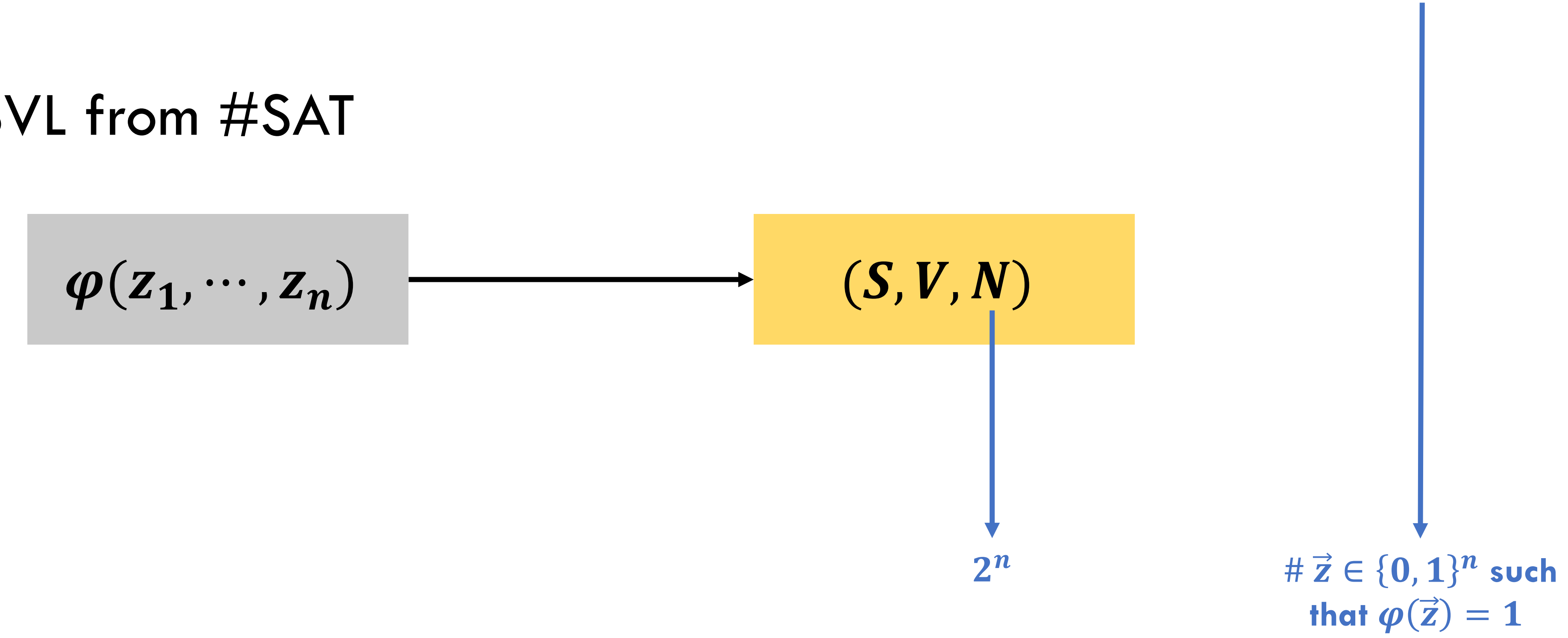
Reduce to SVL from #SAT



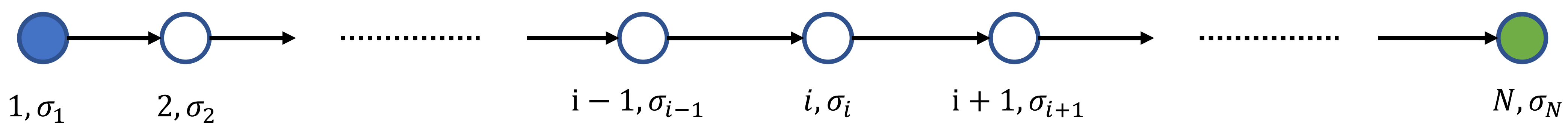
Basic Idea: Long Computation + SNARGs



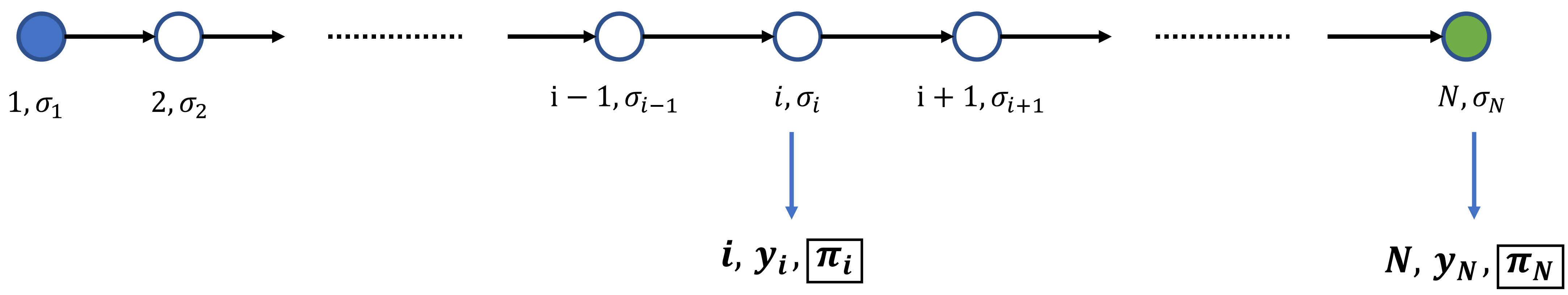
Reduce to SVL from #SAT



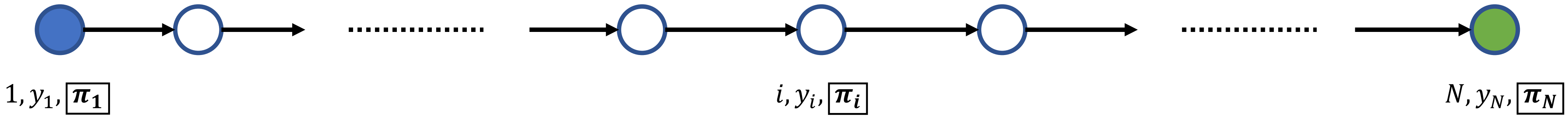
Basic Idea: Long Computation + SNARGs



Basic Idea: Long Computation + SNARGs



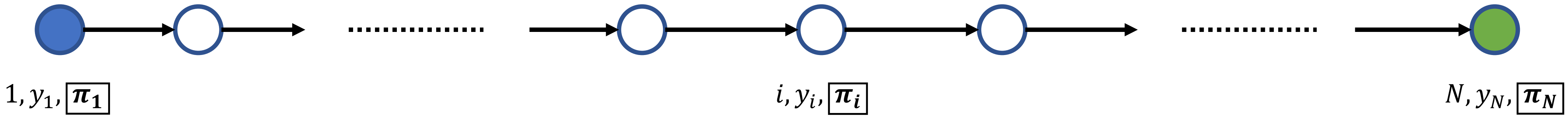
Basic Idea: Long Computation + SNARGs



$\boldsymbol{V}\left(\boldsymbol{i}, \boldsymbol{y_i}, \boxed{\boldsymbol{\pi_i}}\right) = \texttt{ACCEPT} \quad \Longleftrightarrow$

y_i is the # of $\vec{z} \leq i$ such that $\varphi(\vec{z}) = 1$

Basic Idea: Long Computation + SNARGs



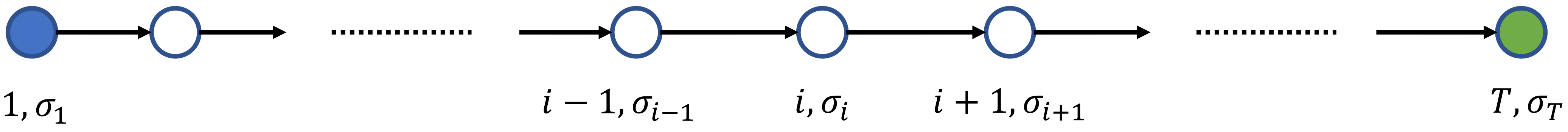
$$S\left(i, y_i, \boxed{\pi_i}\right) = i + 1, y_{i+1}, \boxed{\pi_{i+1}}$$

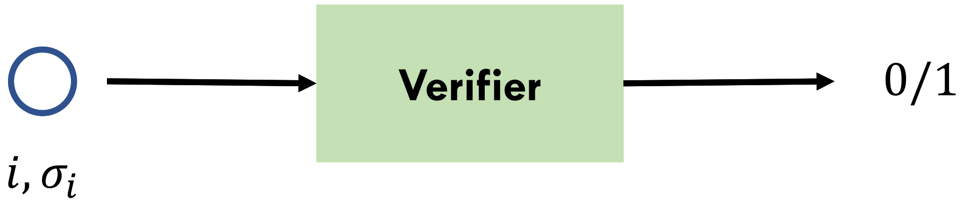
$V\left(i, y_i, \boxed{\pi_i}\right) = \texttt{ACCEPT} \iff$

y_i is the # of $\vec{z} \leq i$ such that $\varphi(\vec{z}) = 1$

Sink of Verifiable Line (SVL)

[Abbott-Kane-Valiant'04,Bitansky-Paneth Rosen'15]





Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

$$\text{Verifier}(i, \sigma_T) = 1$$

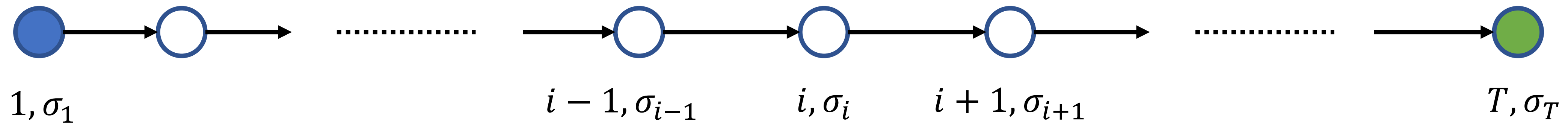
Promise:

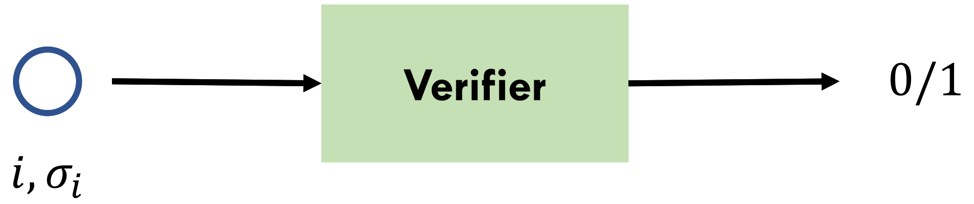
$$\text{Verifier}(i, \sigma_i) = 1 \iff \text{Successor}^{i-1}(1, \sigma_1)$$

SVL **not** in TFNP

Sink of Verifiable Line (SVL)

[Abbott-Kane-Valiant'04,Bitansky-Paneth Rosen'15]





Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

$$\text{Verifier}(i, \sigma_T) = 1$$

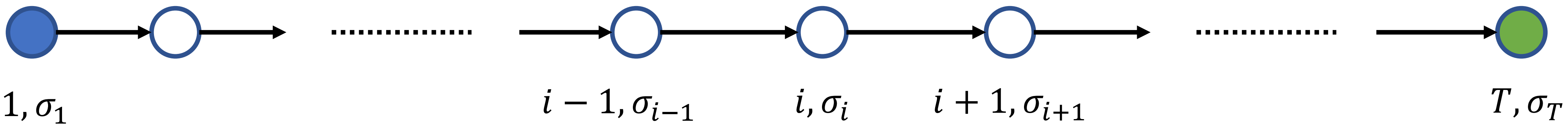
Promise:

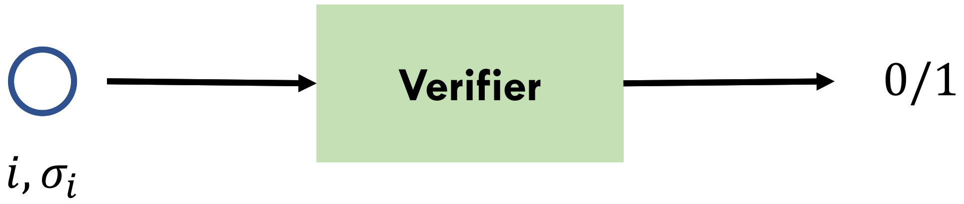
$\text{Verifier}(i, \sigma_i) = 1 \iff \text{Successor}^{i-1}(1, \sigma_1)$

SVL **not** in TFNP

relaxed Sink of Verifiable Line (rSVL)

[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum'19]





Goal: Find (T, σ_T) for $T \in n^{\omega(1)}$ such that

$$\text{Verifier}(i, \sigma_T) = 1$$

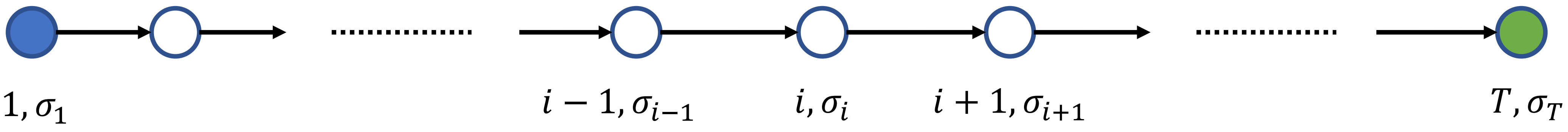
Promise:

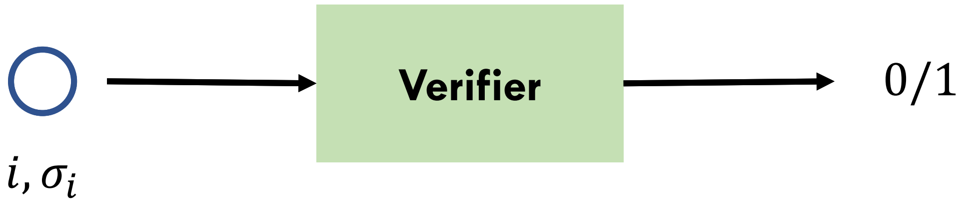
$$\text{Verifier}(i, \sigma_i) = 1 \Leftarrow \text{Successor}^{i-1}(1, \sigma_1)$$

rSVL not in TFNP

relaxed Sink of Verifiable Line (rSVL)

[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum'19]





Goal: For $T \in n^{\omega(1)}$

1. Find (T, σ_T) such that
$$\text{Verifier}(T, \sigma_T) = 1$$
2. Find (i, σ) such that $(i, \sigma_i) \neq \text{Successor}^{i-1}(1, \sigma_1)$ but
$$\text{Verifier}(i, \sigma_i) = 1$$

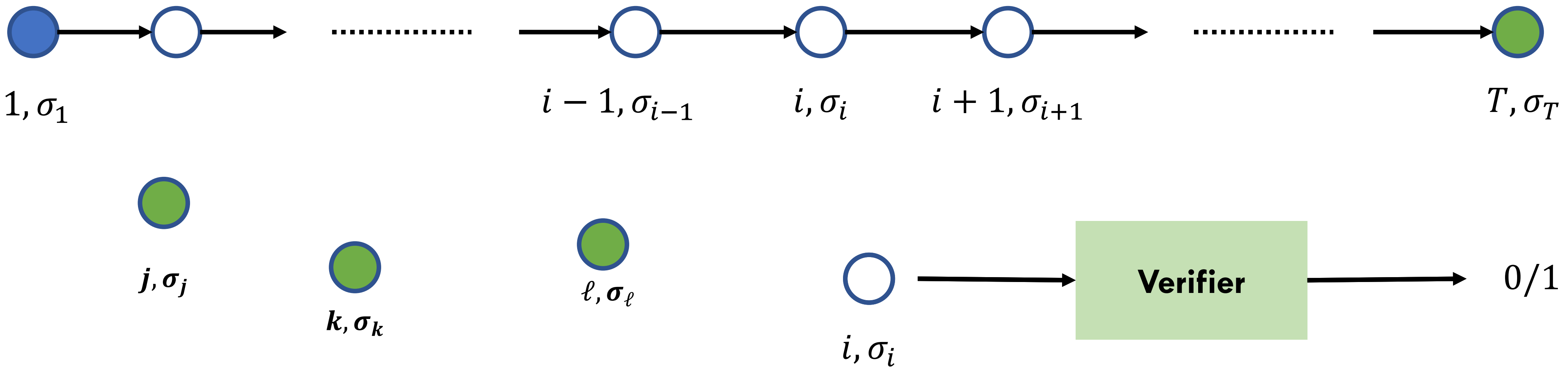
Promise:

$$\text{Verifier}(i, \sigma_i) = 1 \Leftarrow \text{Successor}^{i-1}(1, \sigma_1)$$

rSVL **not** in TFNP

relaxed Sink of Verifiable Line (rSVL)

[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum'19]



Goal: For $T \in n^{\omega(1)}$

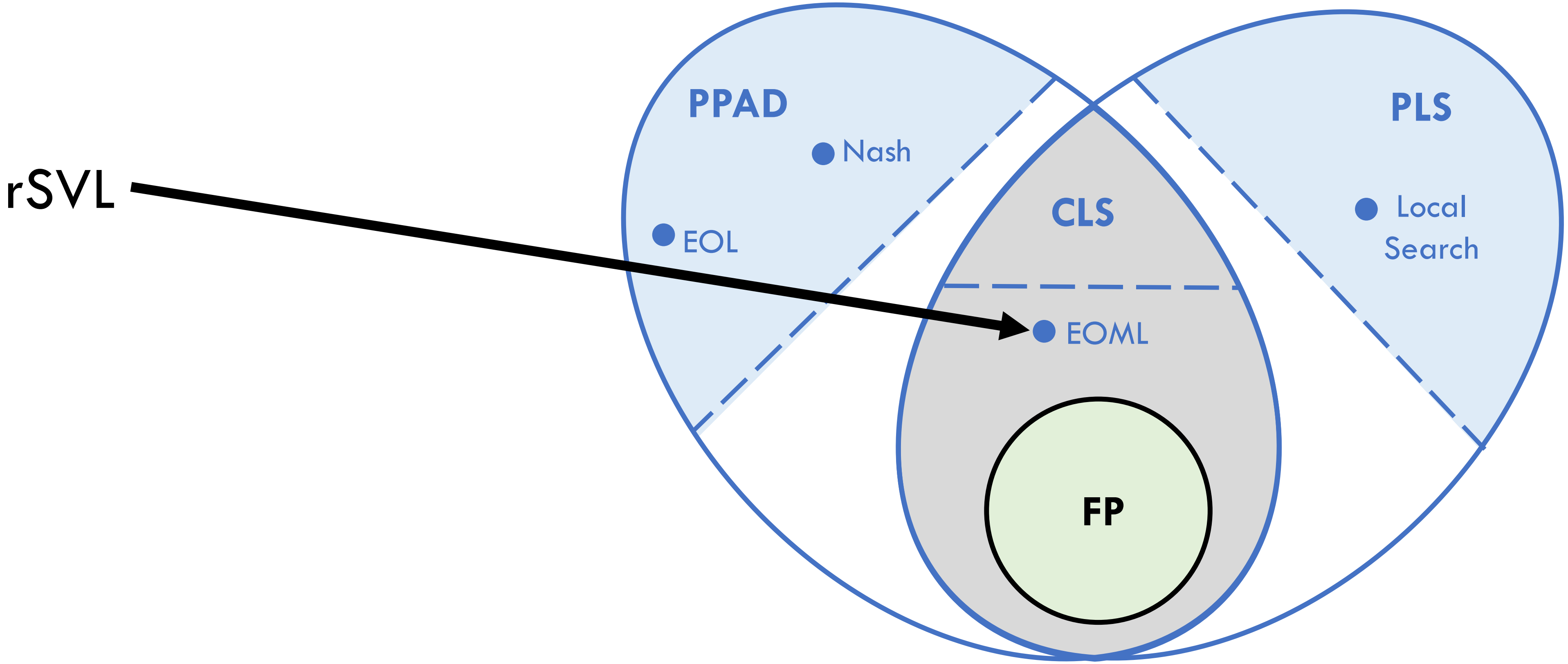
1. Find (T, σ_T) such that
   $$\text{Verifier}(T, \sigma_T) = 1$$
2. Find (i, σ) such that $(i, \sigma_i) \neq \text{Successor}^{i-1}(1, \sigma_1)$ but
   $$\text{Verifier}(i, \sigma_i) = 1$$

Promise:

$$\text{Verifier}(i, \sigma_i) = 1 \Leftarrow \text{Successor}^{i-1}(1, \sigma_1)$$

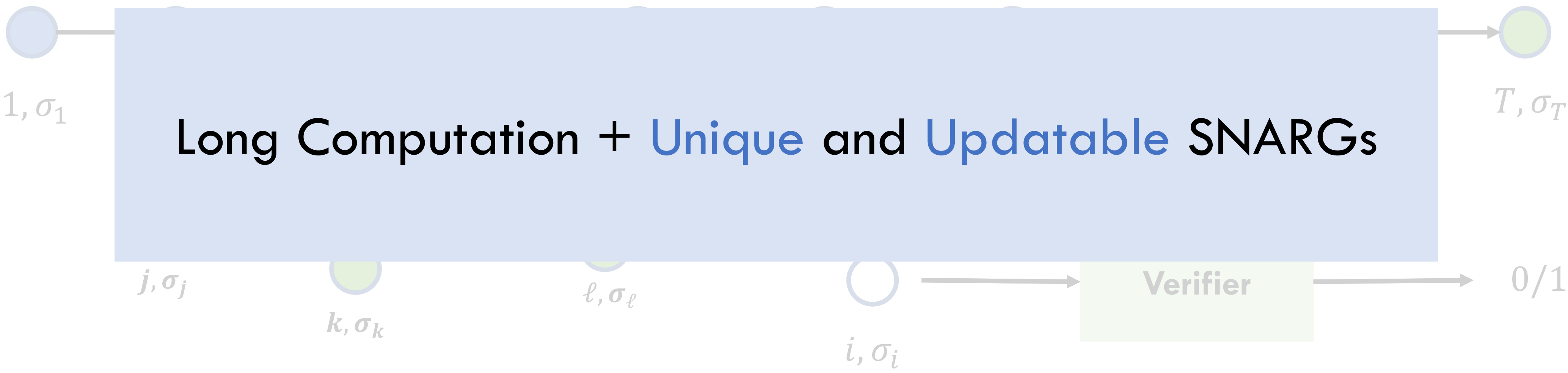
rSVL not in TFNP

rSVL Reduces to EOML



relaxed Sink of Verifiable Line (rSVL)

[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum'19]



Long Computation + Unique and Updatable SNARGs

Goal: For $T \in n^{\omega(1)}$

1. Find (T, σ_T) such that
   $\text{Verifier}(T, \sigma_T) = 1$
2. Find (i, σ) such that $(i, \sigma_i) \neq \text{Successor}^{i-1}(1, \sigma_1)$ but
   $\text{Verifier}(i, \sigma_i) = 1$

Promise:

$\text{Verifier}(i, \sigma_i) = 1 \Leftarrow \text{Successor}^{i-1}(1, \sigma_1)$

rSVL not in TFNP

Arithmetization of SAT

Arithmetization

$\varphi(z_1, \cdots, z_n)$ → $p\colon \mathbb{F}^n \to \mathbb{F}$

$\text{degree}(p) = d = 4$

#SAT ≤ #3SAT4

Arithmetization of SAT

Arithmetization

$\varphi(z_1, \cdots, z_n)$ → $p\colon \mathbb{F}^n \to \mathbb{F}$

↓

$\text{degree}(p) = d = 4$

#SAT ≤ #3SAT4

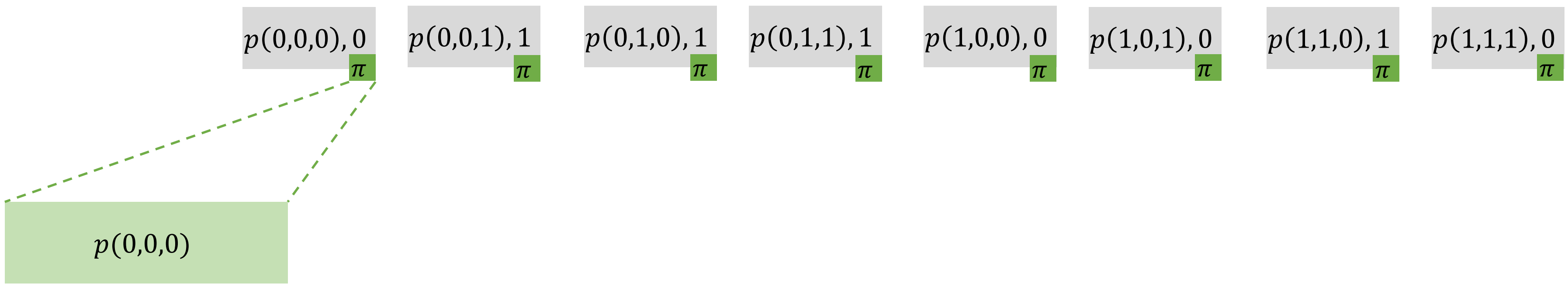
Number of $\vec{z} \in \{0,1\}^n$ such that $\varphi(\vec{z}) = 1$ is

$$y = \sum_{\vec{z} \in \{0,1\}^n} p(\vec{z})$$

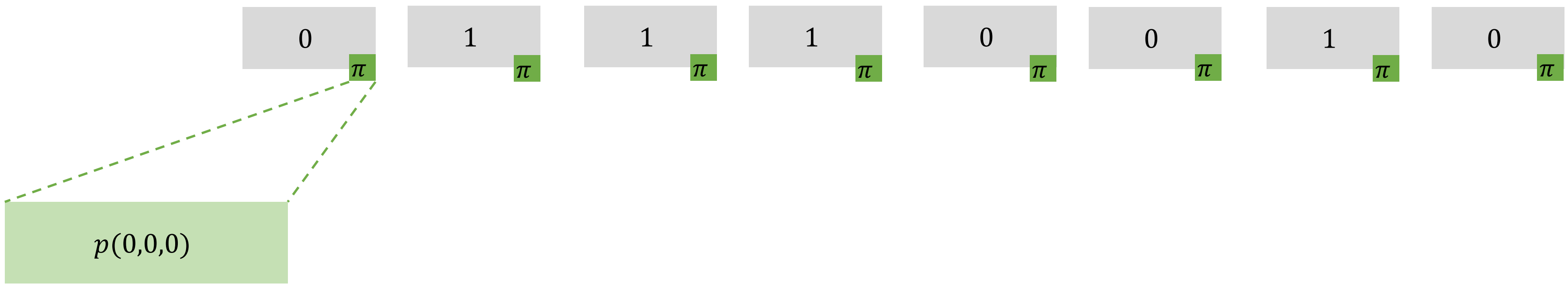
Verifiable Aggregation of SAT solutions

$p(0,0,0), 0$ $p(0,0,1), 1$ $p(0,1,0), 1$ $p(0,1,1), 1$ $p(1,0,0), 0$ $p(1,0,1), 0$ $p(1,1,0), 1$ $p(1,1,1), 0$

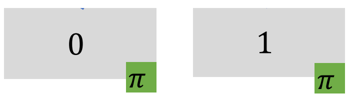
Verifiable Aggregation of SAT solutions



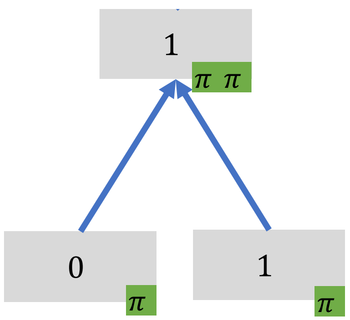
Verifiable Aggregation of SAT solutions



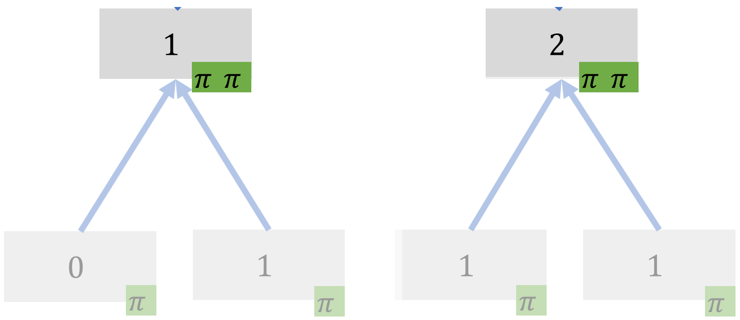
Verifiable Aggregation of SAT solutions



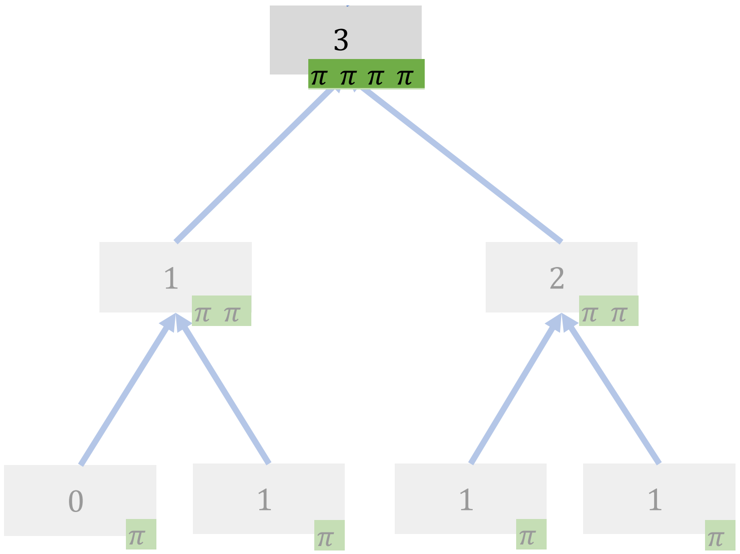
Verifiable Aggregation of SAT solutions



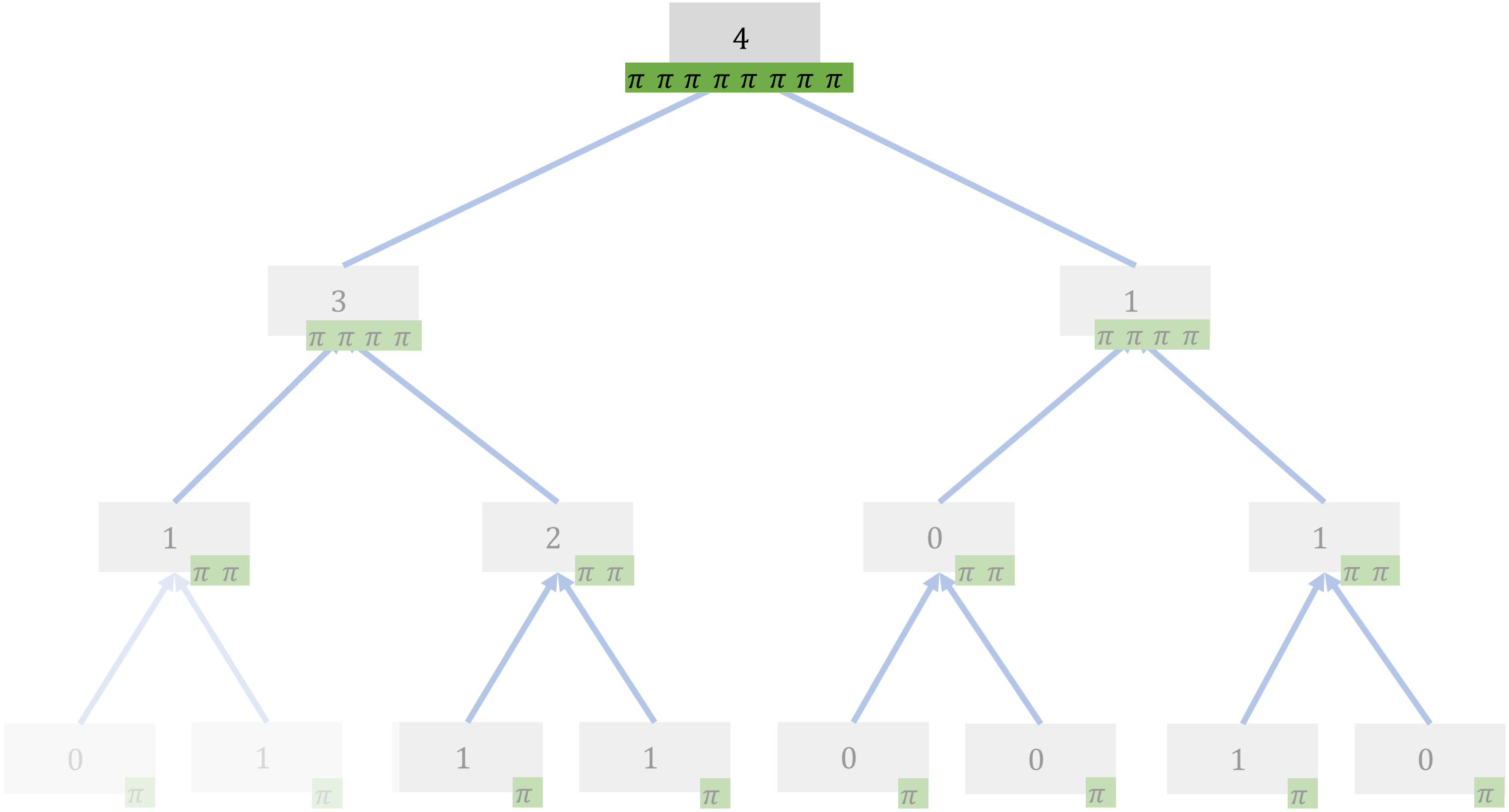
Verifiable Aggregation of SAT solutions



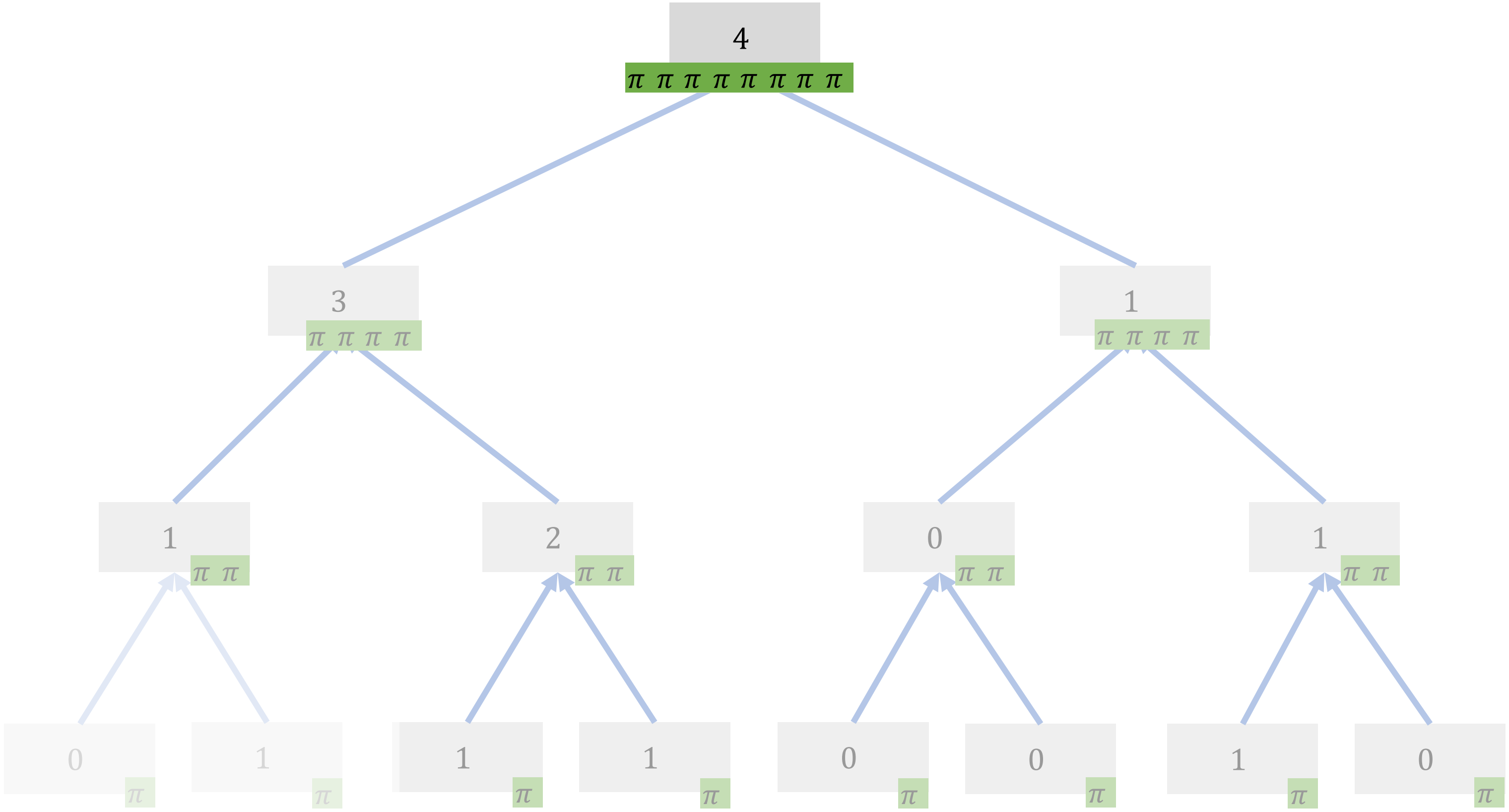
Verifiable Aggregation of SAT solutions



Verifiable Aggregation of SAT solutions



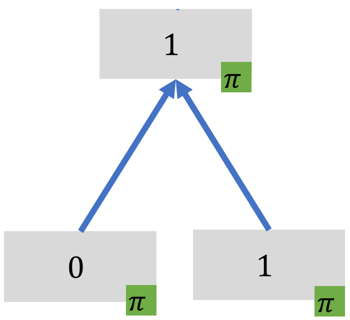
Verifiable Aggregation of SAT solutions



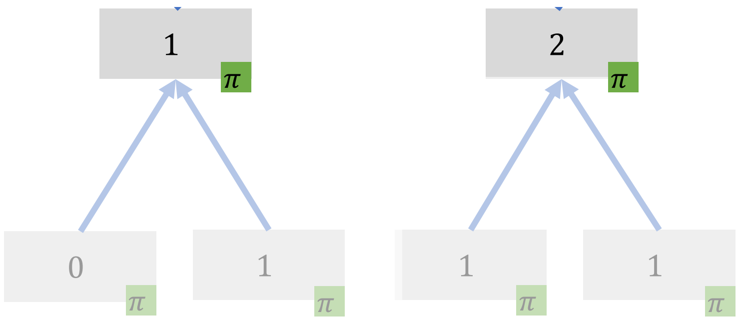
For $T = 2^n$, number of proofs is $O(T)$!

Proof Merge [Valiant'06]

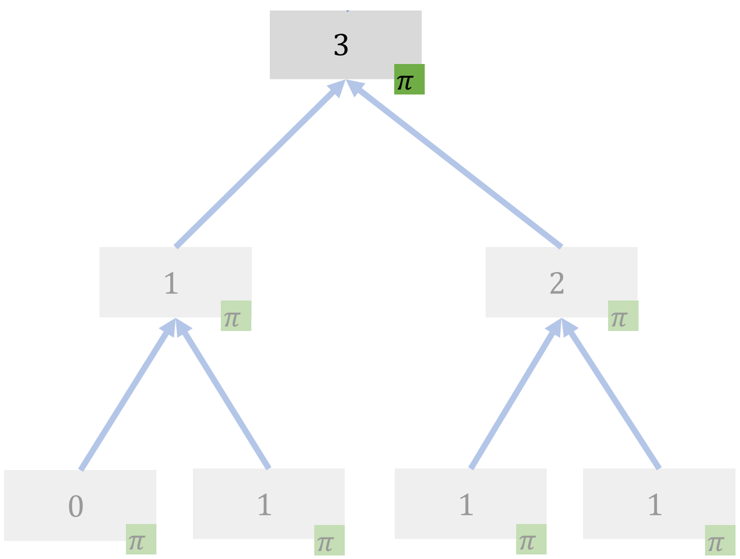
Merge proofs into a single proof in $\mathrm{poly}(n)$ time



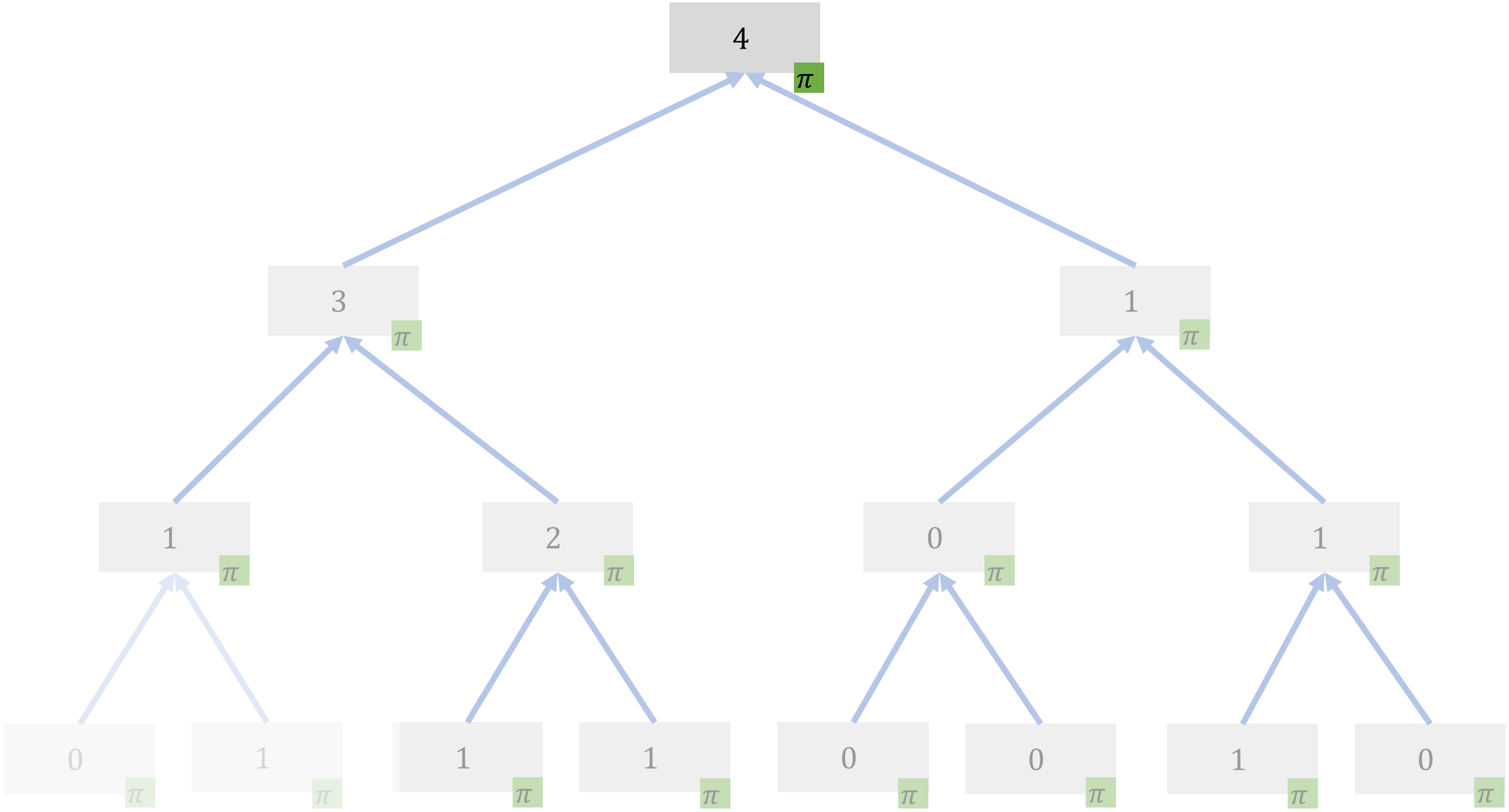
Proof Merge [Valiant'06]



Proof Merge [Valiant'06]

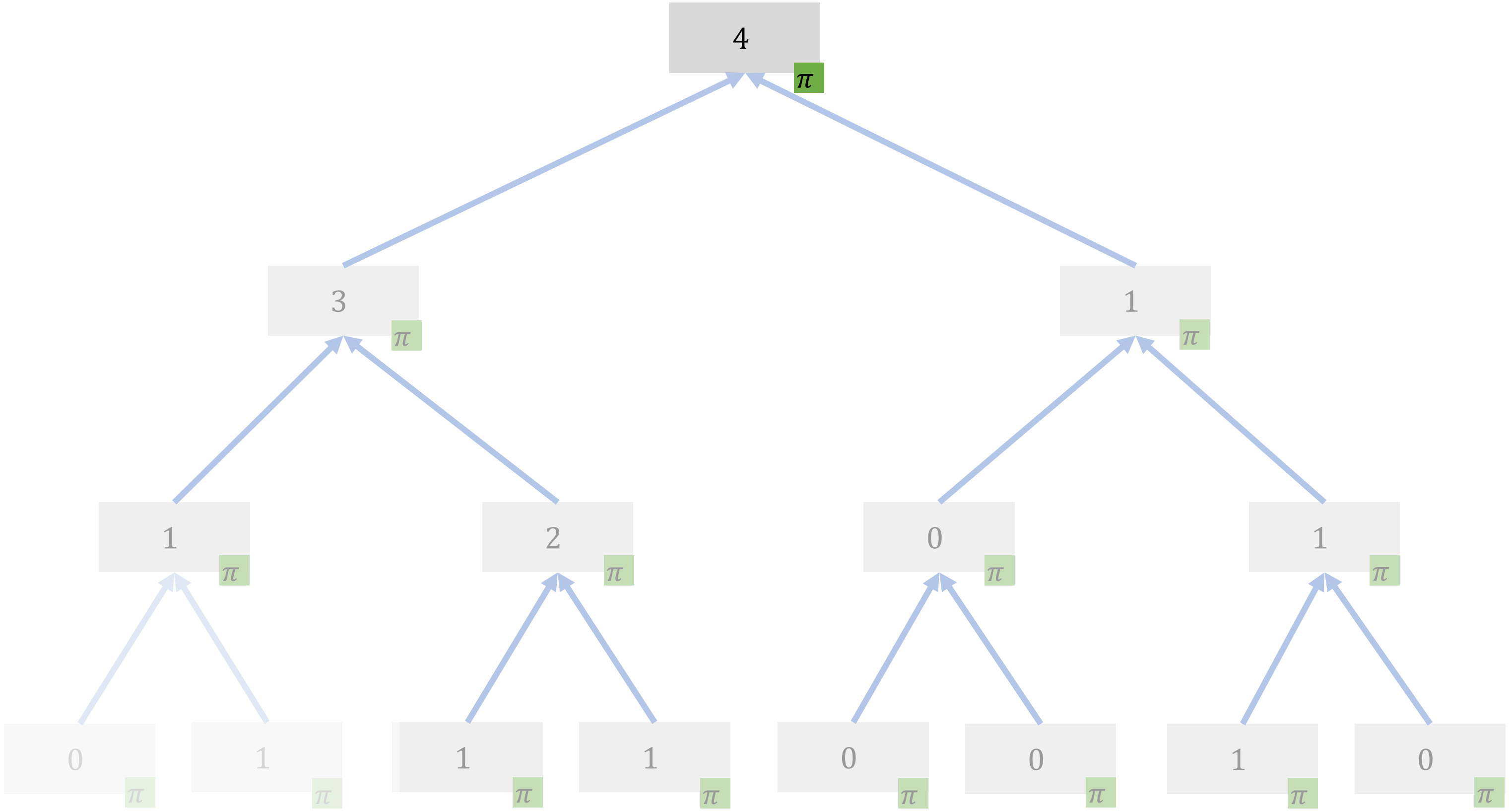


Proof Merge [Valiant'06]



For $T = 2^n$, number of proofs is $O(1)$!

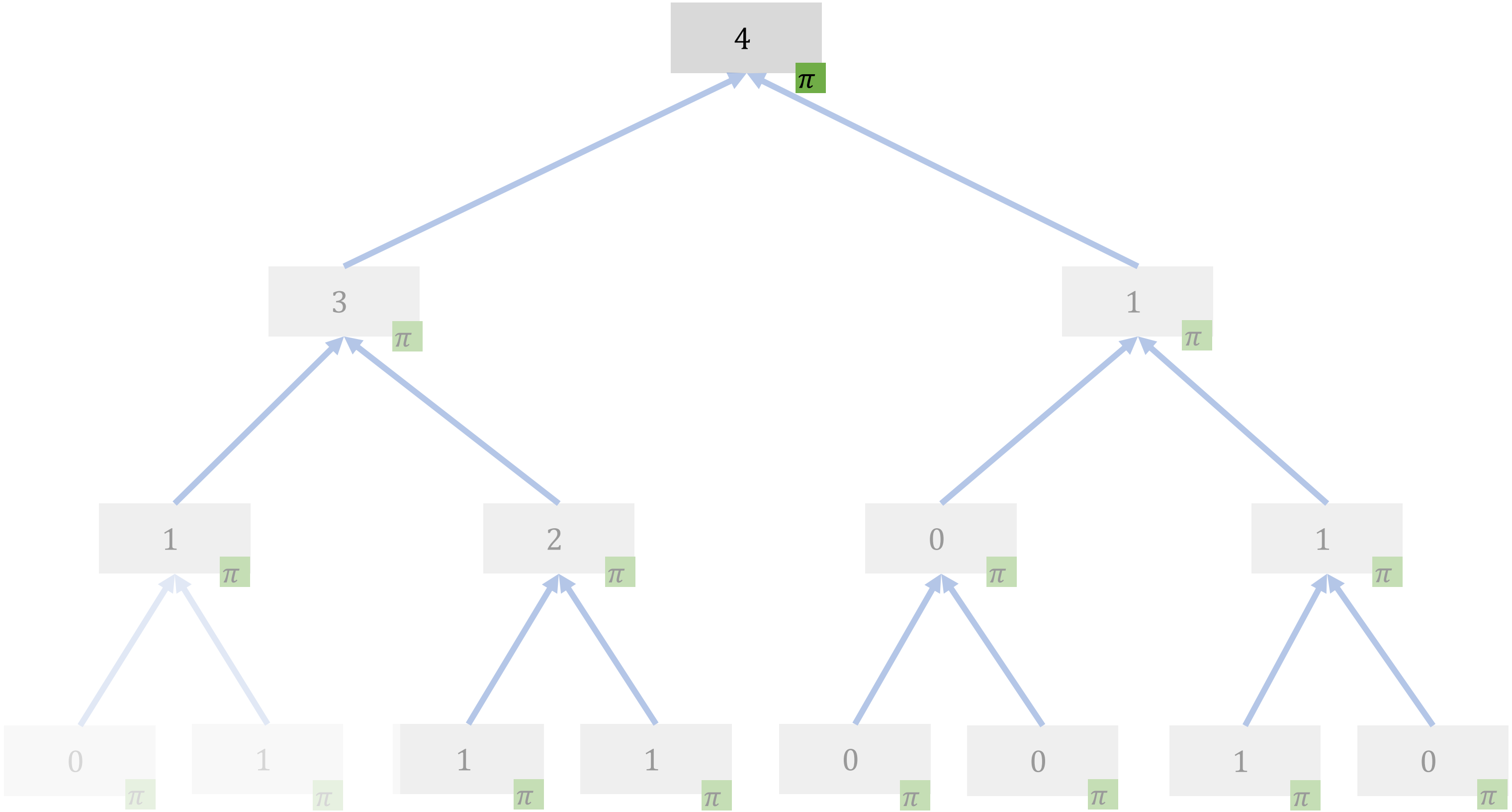
Proof Merge [Valiant'06]



For $T = 2^n$, number of proofs is $O(1)$!

Non-standard assumptions.

Proof Merge [Valiant'06]



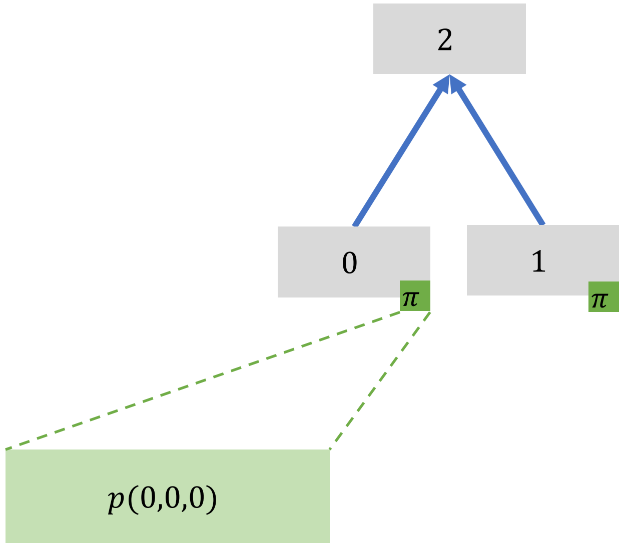
For $T = 2^n$, number of proofs is $O(1)$!

Non-standard assumptions.

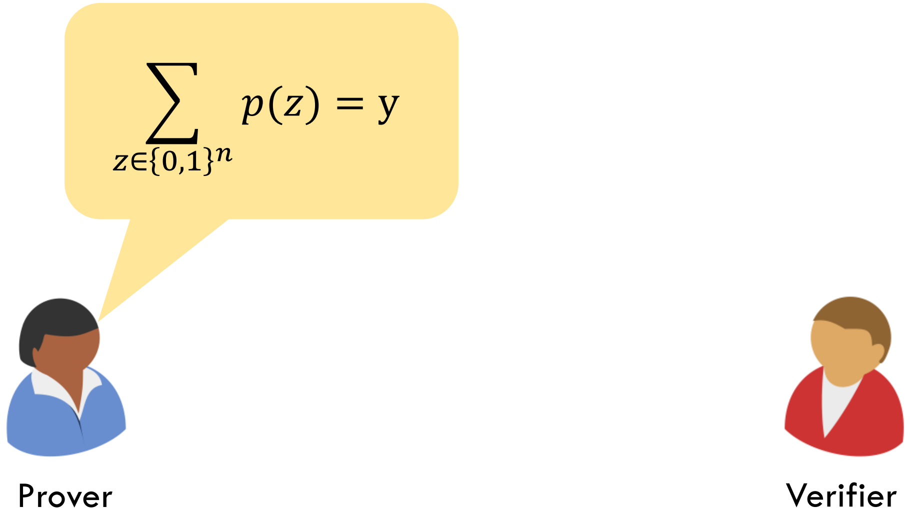
Proofs not computationally unique.

Incremental Merge

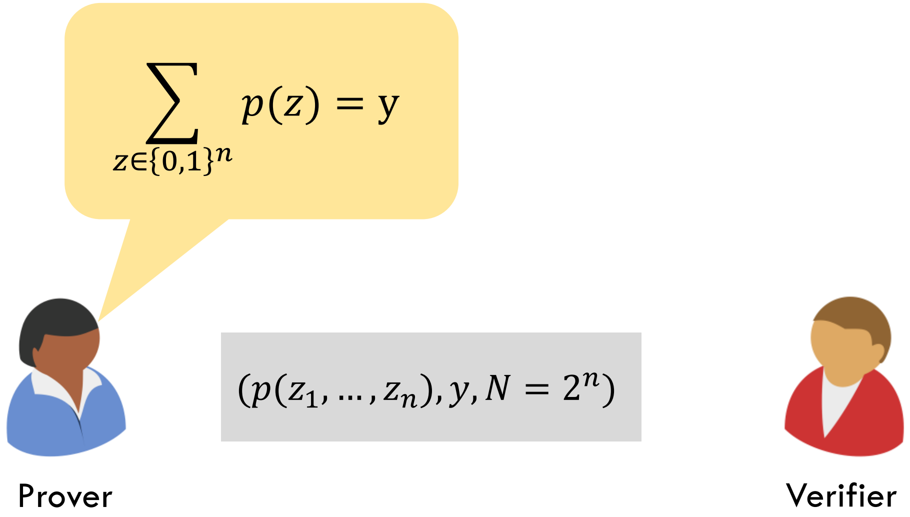
[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum’19]



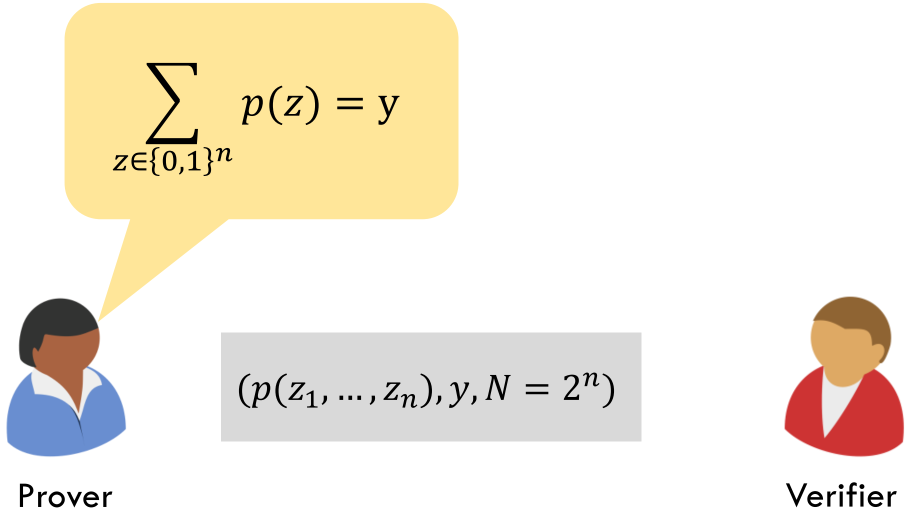
Sumcheck [Lund-Fortnow-Karloff-Nisan'06]



Sumcheck [Lund-Fortnow-Karloff-Nisan'06]



Sumcheck [Lund-Fortnow-Karloff-Nisan'06]



$$p_1(X) = \sum_{z_2, \ldots, z_n \in \{0,1\}} p(X, z_2, \ldots, z_n)$$

$p_1(X)$ →

Sumcheck [Lund-Fortnow-Karloff-Nisan'06]

$$\sum_{z \in \{0,1\}^n} p(z) = y$$

Prover

$(p(z_1, \ldots, z_n), y, N = 2^n)$

Verifier

$$p_1(X) = \sum_{z_2, \ldots, z_n \in \{0,1\}} p(X, z_2, \ldots, z_n)$$

$p_1(0), p_1(1), \cdots, p_1(d)$ →

Sumcheck [Lund-Fortnow-Karloff-Nisan'06]

$$\sum_{z \in \{0,1\}^n} p(z) = y$$

Prover

$(p(z_1, \ldots, z_n), y, N = 2^n)$

Verifier

$$p_1(X) = \sum_{z_2, \ldots, z_n \in \{0,1\}} p(X, z_2, \ldots, z_n)$$

$p_1(0), p_1(1), \cdots, p_1(d)$ →

$p_1(0) + p_1(1) \stackrel{?}{=} y$

$\beta_1 \leftarrow_R \mathbb{F}$

← β_1

Sumcheck [Lund-Fortnow-Karloff-Nisan'06]

$$\sum_{z \in \{0,1\}^n} p(z) = y$$

Prover

Verifier

$(p(z_1, \ldots, z_n), y, N = 2^n)$

$$p_1(X) = \sum_{z_2, \ldots, z_n \in \{0,1\}} p(X, z_2, \ldots, z_n)$$

$p_1(0), p_1(1), \cdots, p_1(d)$ →

$p_1(0) + p_1(1) \stackrel{?}{=} y$

$\beta_1 \leftarrow_R \mathbb{F}$

← β_1

$y_1 = p_1(\beta_1)$

$y_1 = p_1(\beta_1)$

$(p(\beta_1, \ldots, z_n), y_1, N/2)$

Sumcheck [Lund-Fortnow-Karloff-Nisan'06]

$$\sum_{z \in \{0,1\}^n} p(z) = y$$

Prover

Verifier

$(p(z_1, \ldots, z_n), y, N = 2^n)$

$$p_1(X) = \sum_{z_2,\ldots,z_n \in \{0,1\}} p(X, z_2, \ldots, z_n)$$

$p_1(0), p_1(1), \cdots, p_1(d)$ →

$p_1(0) + p_1(1) \stackrel{?}{=} y$

$\beta_1 \leftarrow_R \mathbb{F}$

← β_1

$y_1 = p_1(\beta_1)$ $\qquad$ $y_1 = p_1(\beta_1)$

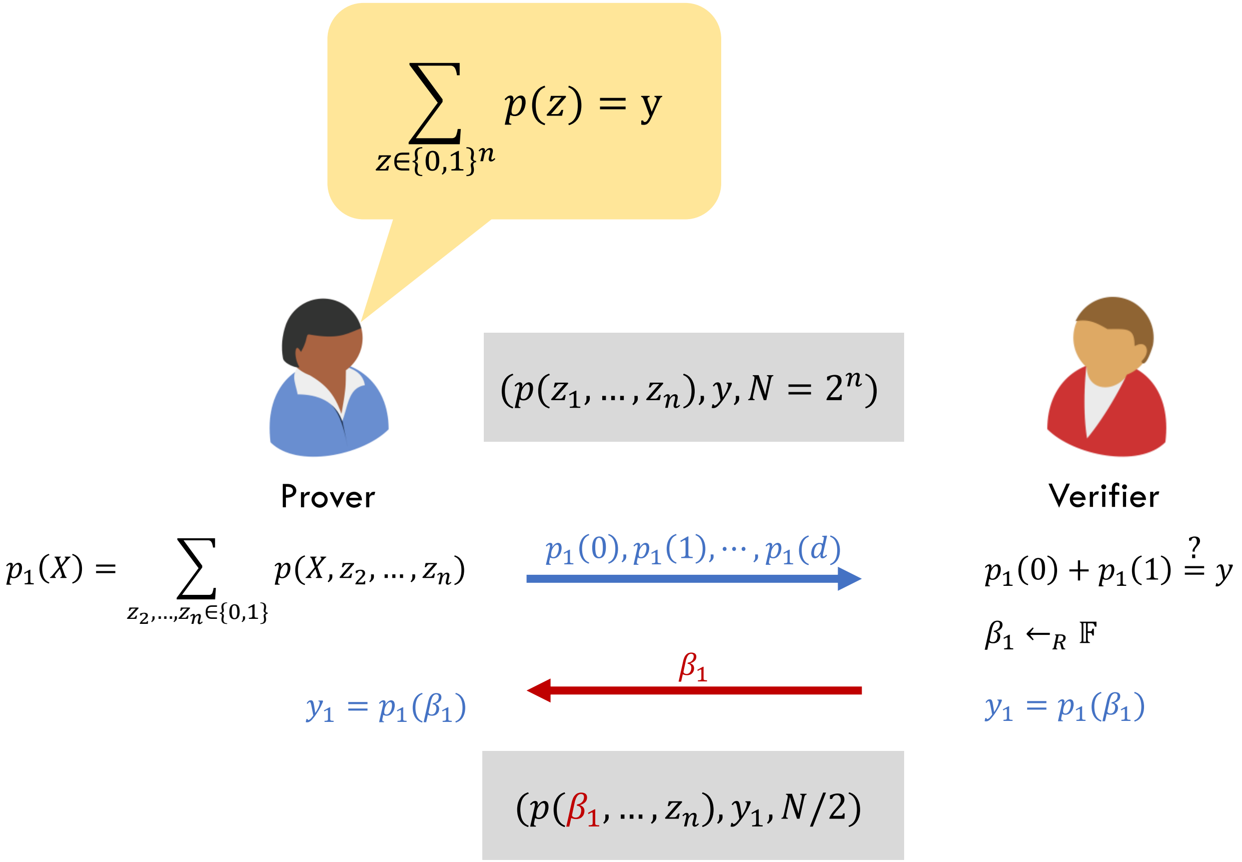
$(p(\beta_1, \ldots, z_n), y_1, N/2)$

Sound and Unique

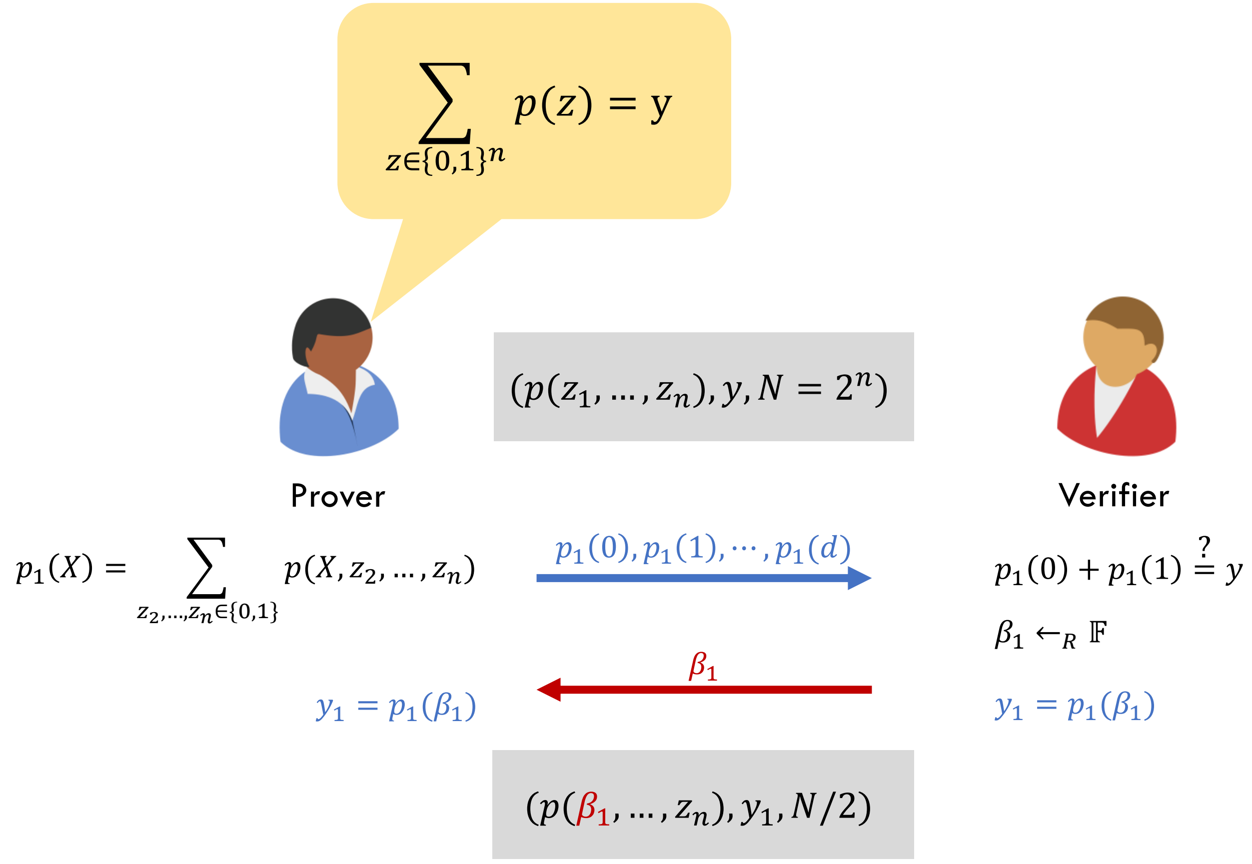
Sumcheck [Lund-Fortnow-Karloff-Nisan'06]

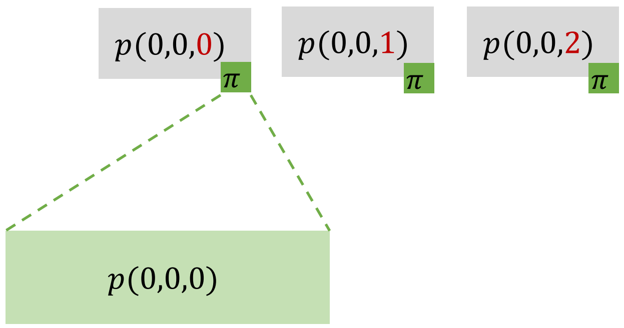
Outline-and-Batch [Bitansky-C-Holmgren-Kamath-Lombardi-Paneth-Rothblum'22]

1. Downward self reduction to $d+1$ statements of size $N/2$.
2. Batch $d+1$ statements into a single randomized statement of size $N/2$ using verifier randomness β.

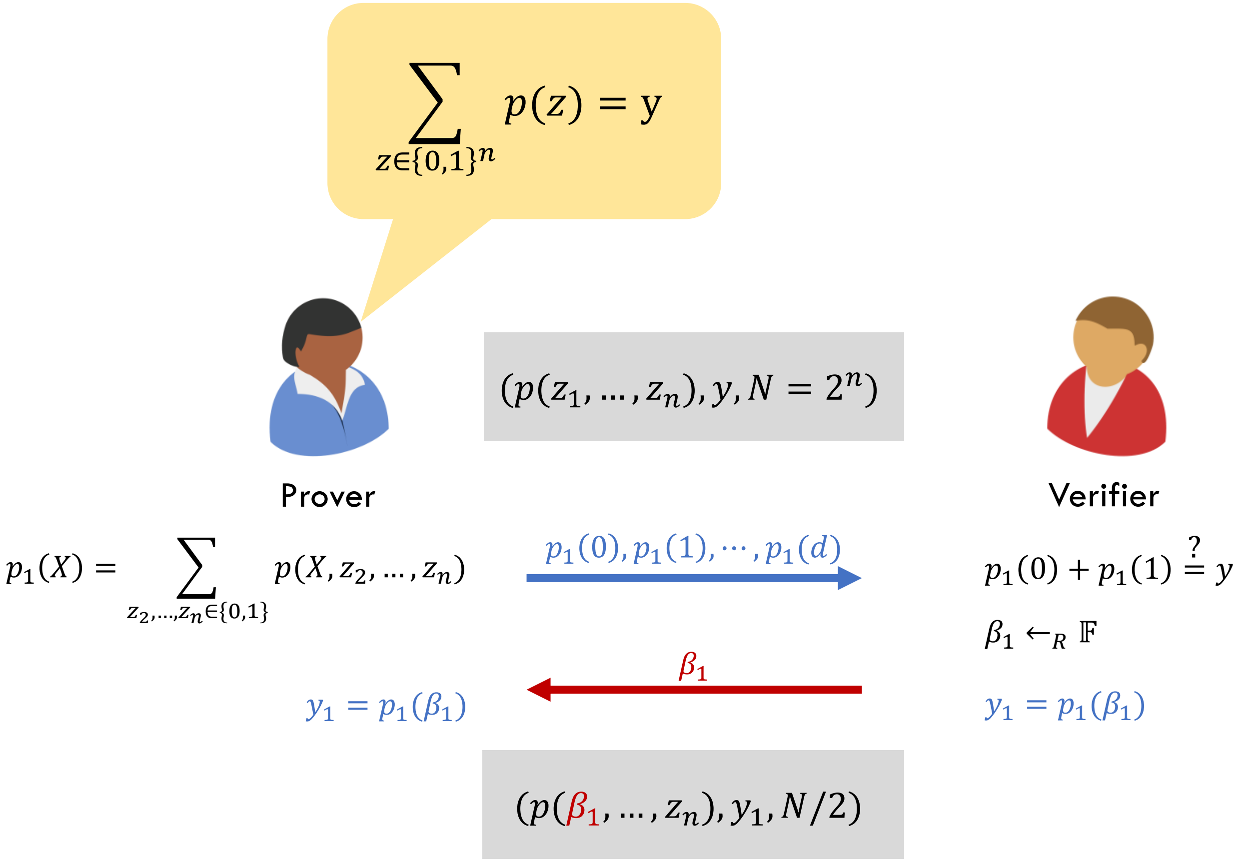


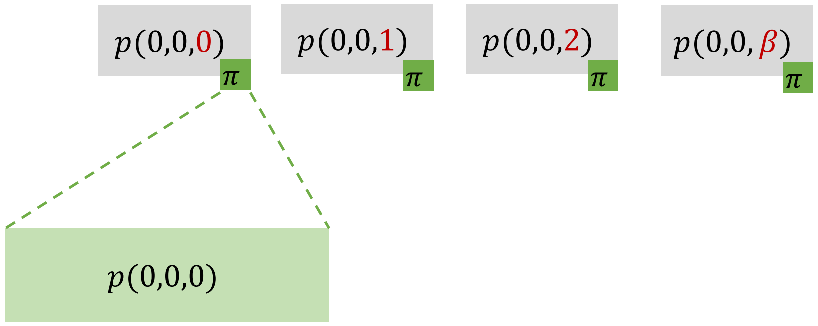
Incremental Merge



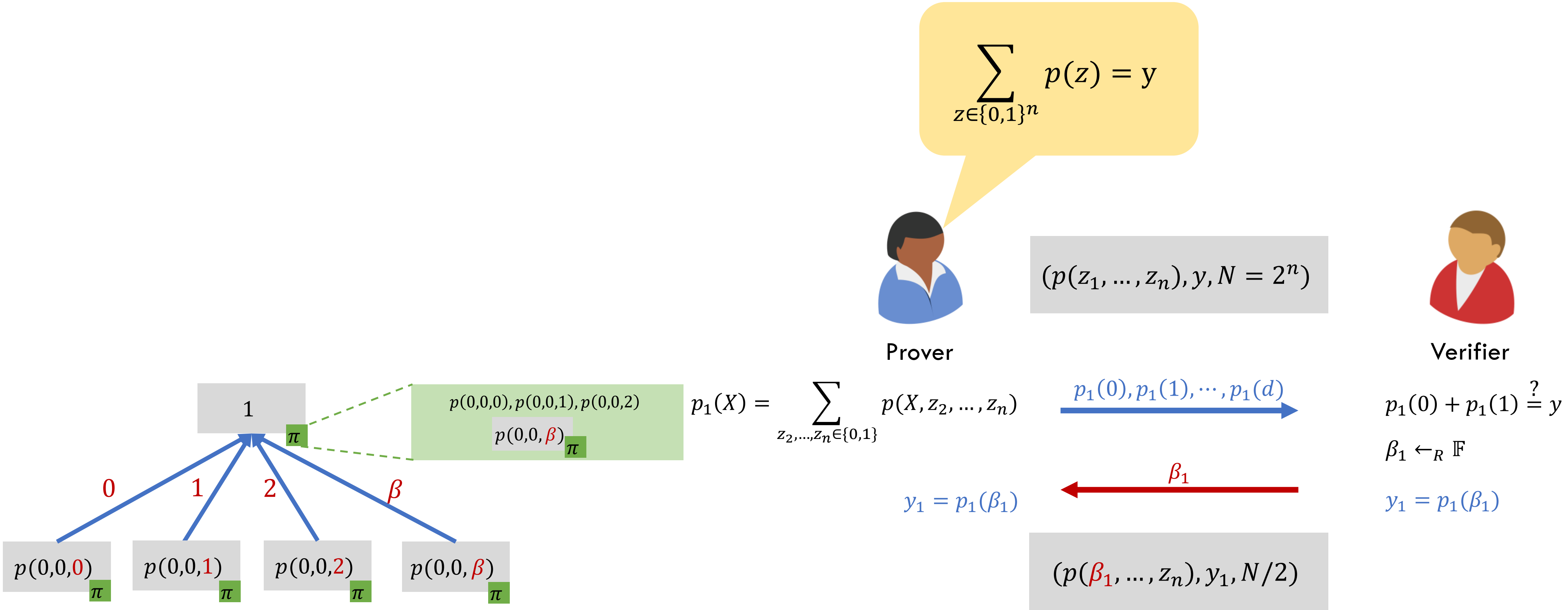


Incremental Merge

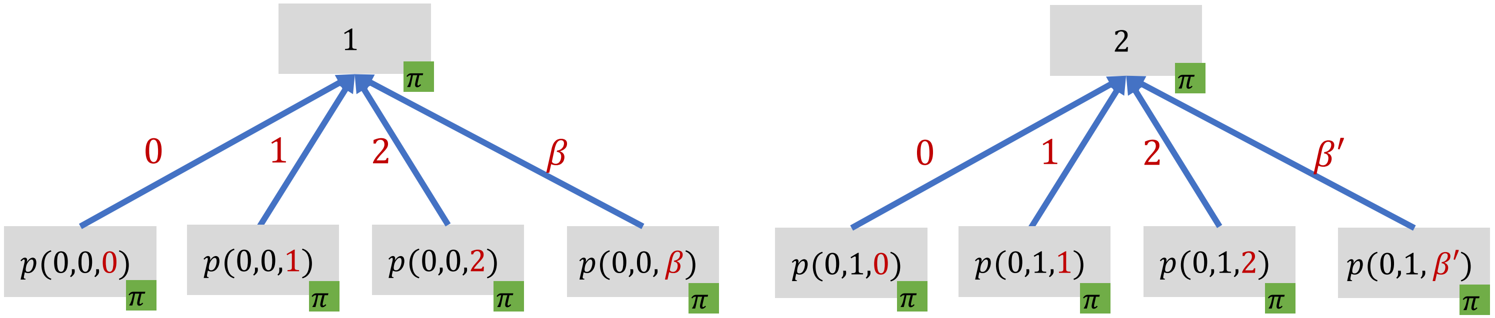




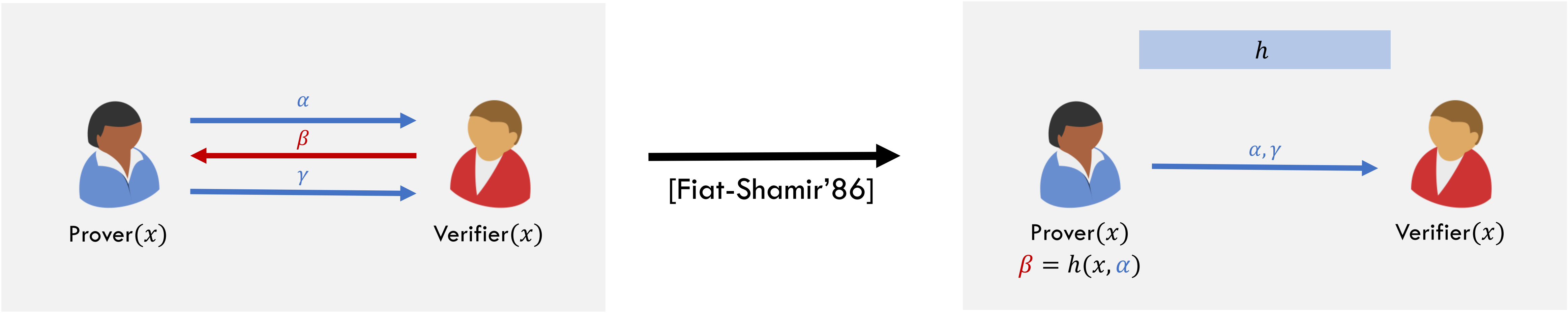
Incremental Merge



Incremental Merge



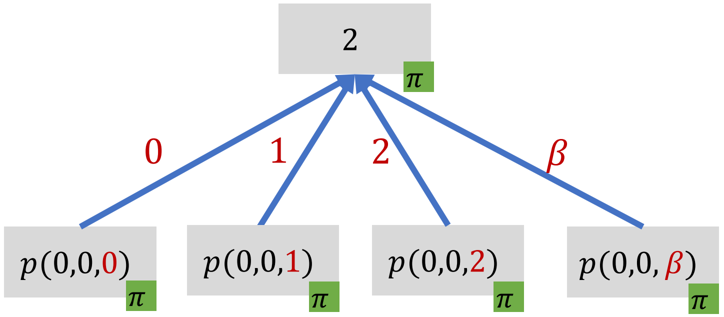
Fiat-Shamir (FS) Methodology

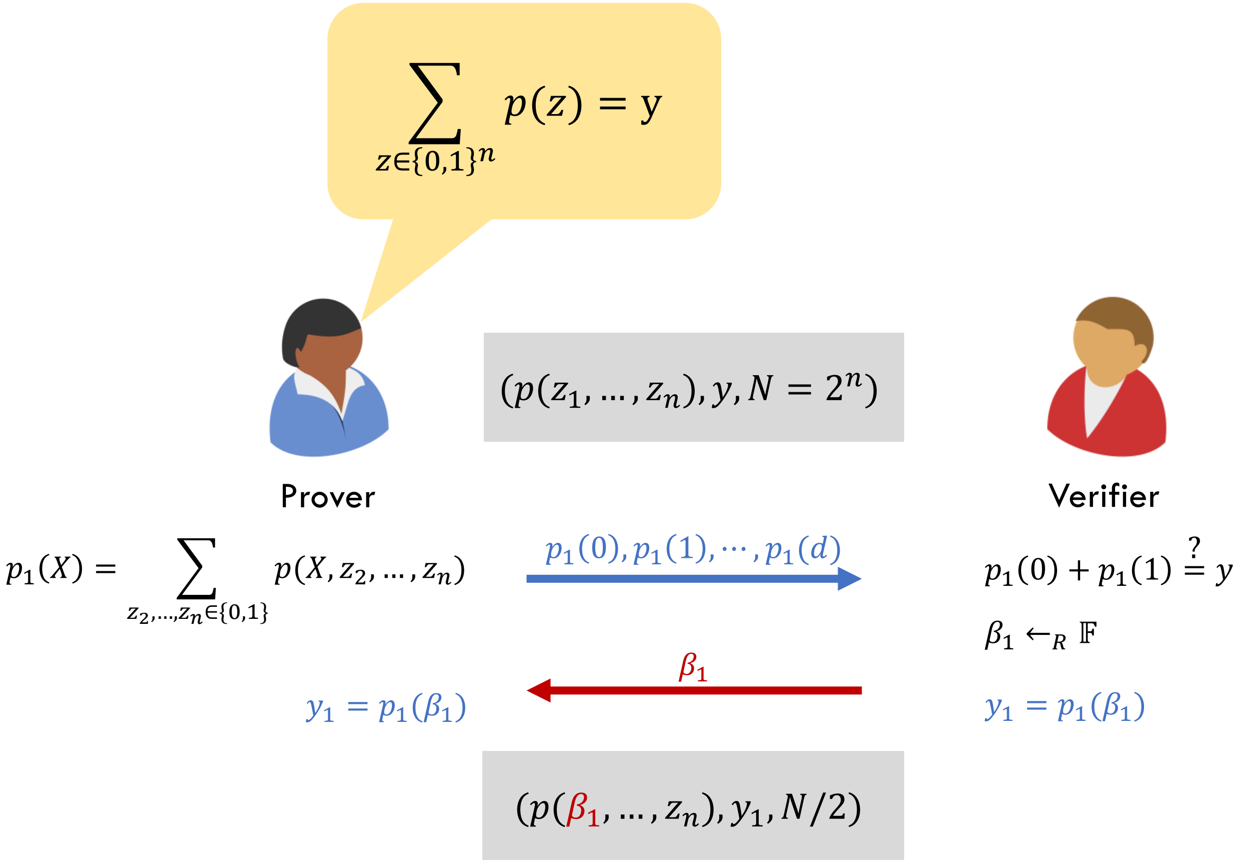


β is a random string

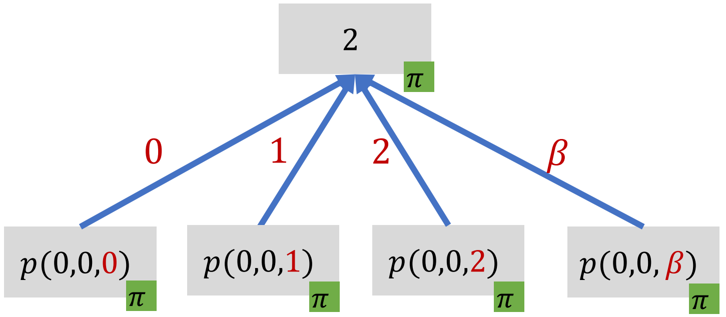
Assumption: There exists a hash function such that the transformation is sound.

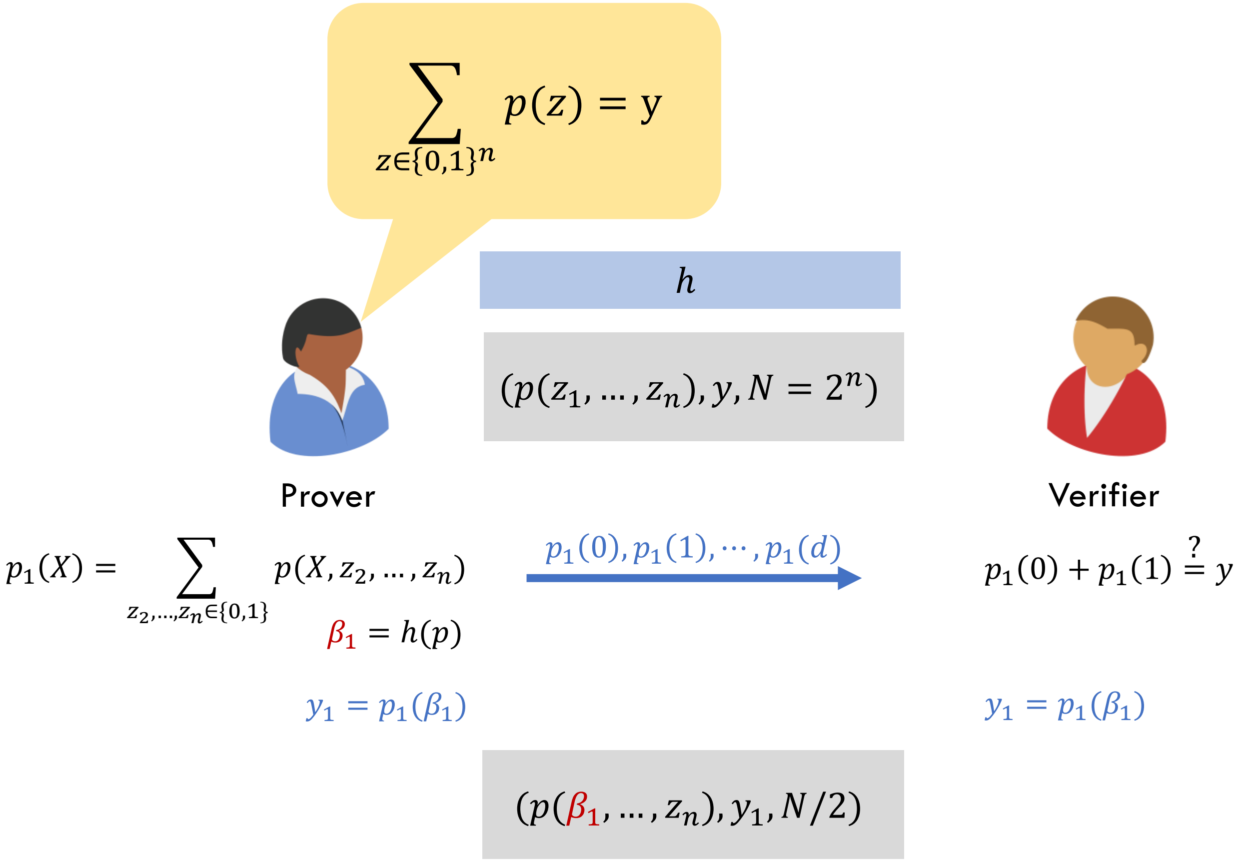
Incremental Merge





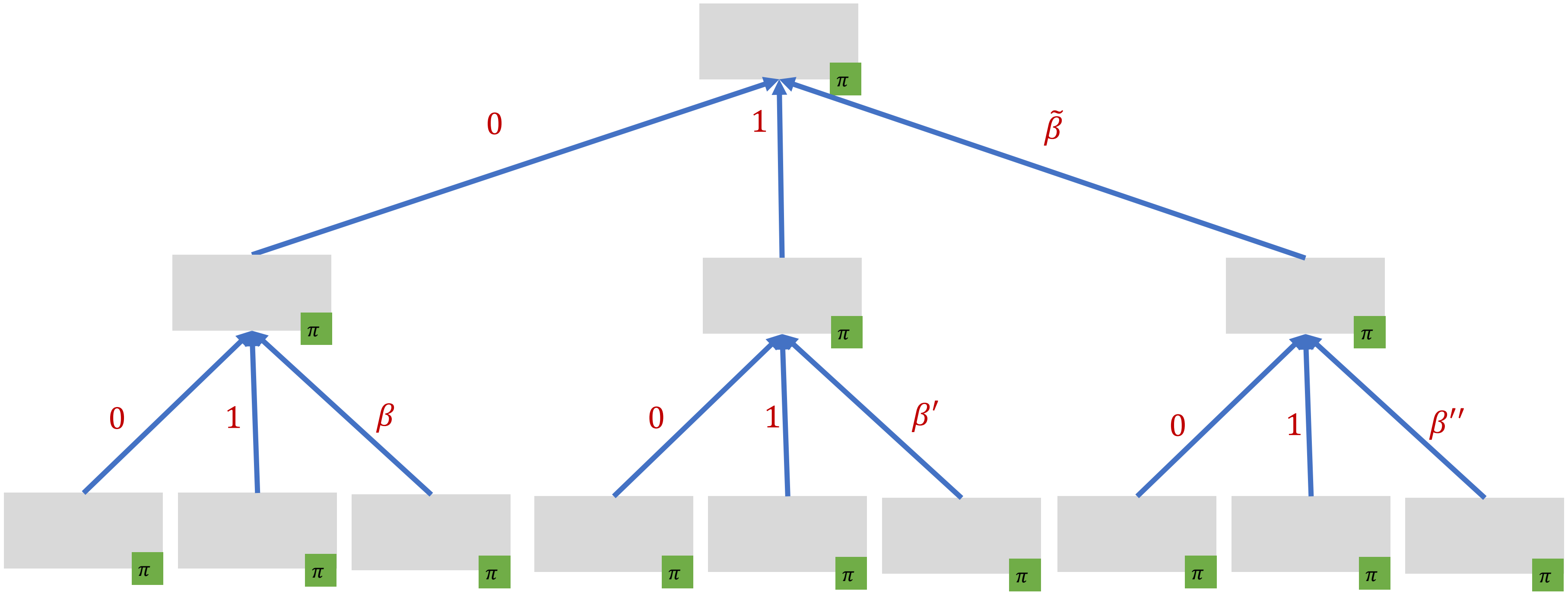
Incremental Merge



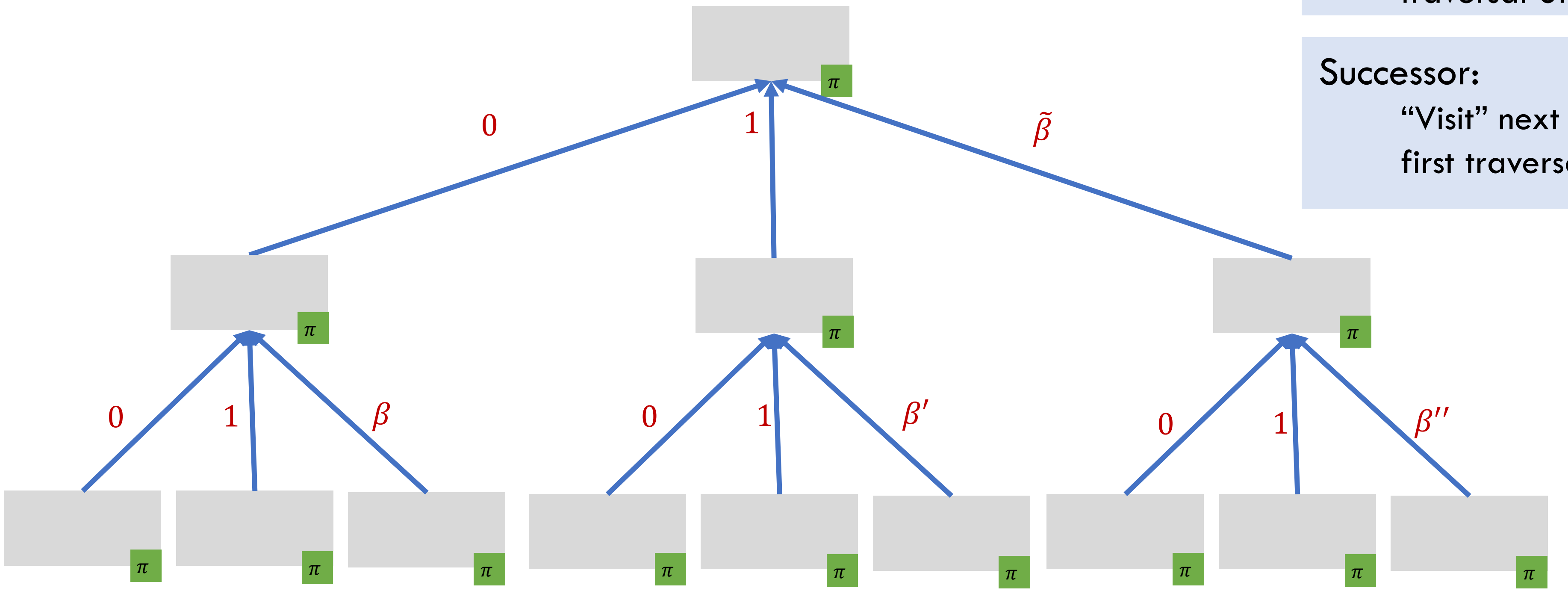


By Fiat-Shamir, the randomized reduction to smaller instance is non-interactive.

rSVL Labels



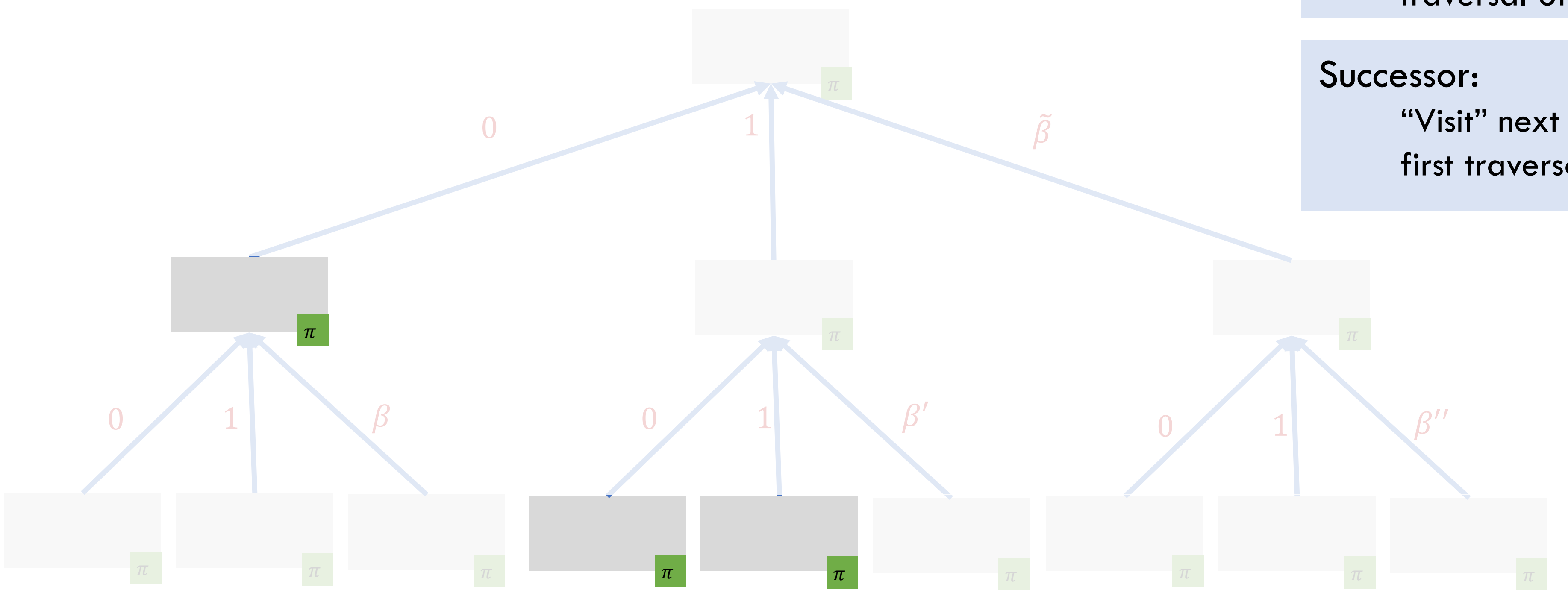
rSVL Labels



rSVL Labels:
Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:
"Visit" next node on the depth first traversal.

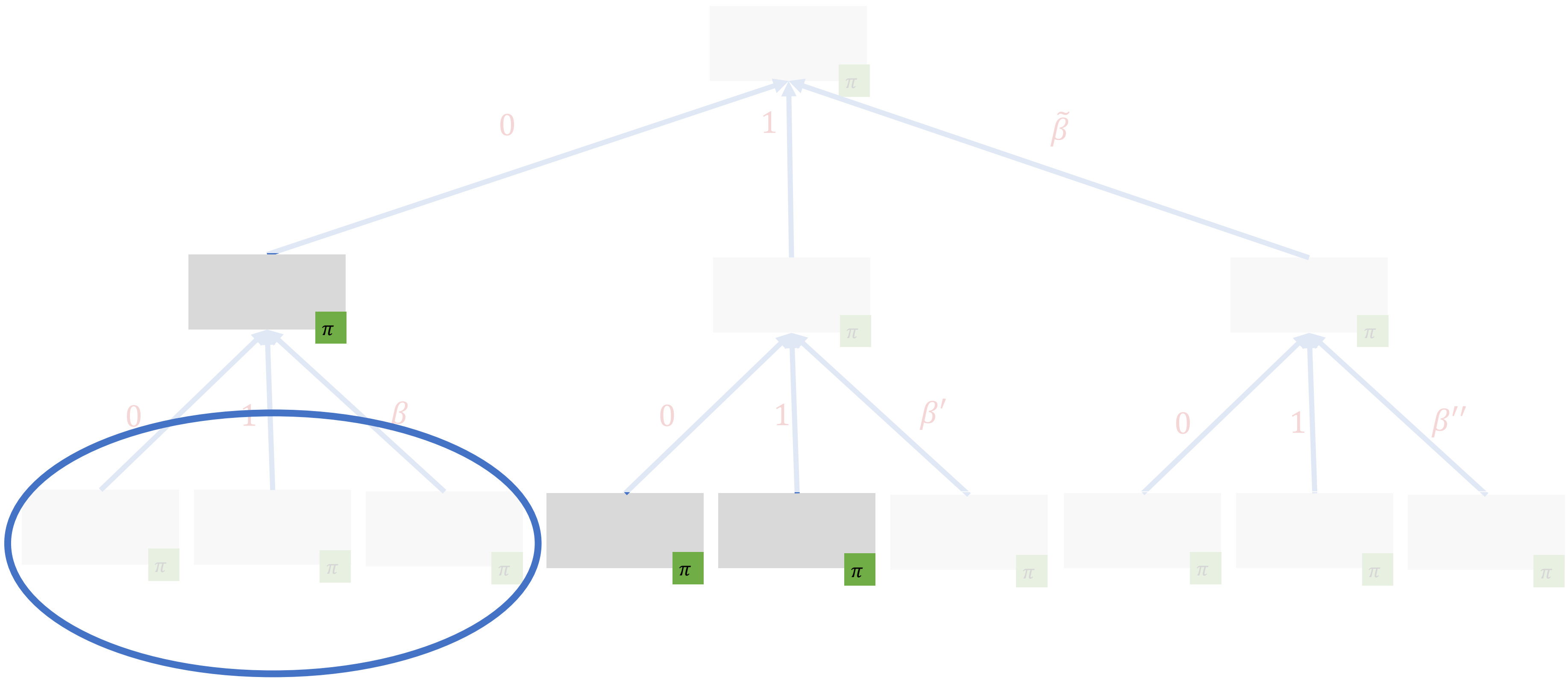
rSVL Labels



rSVL Labels:
Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:
"Visit" next node on the depth first traversal.

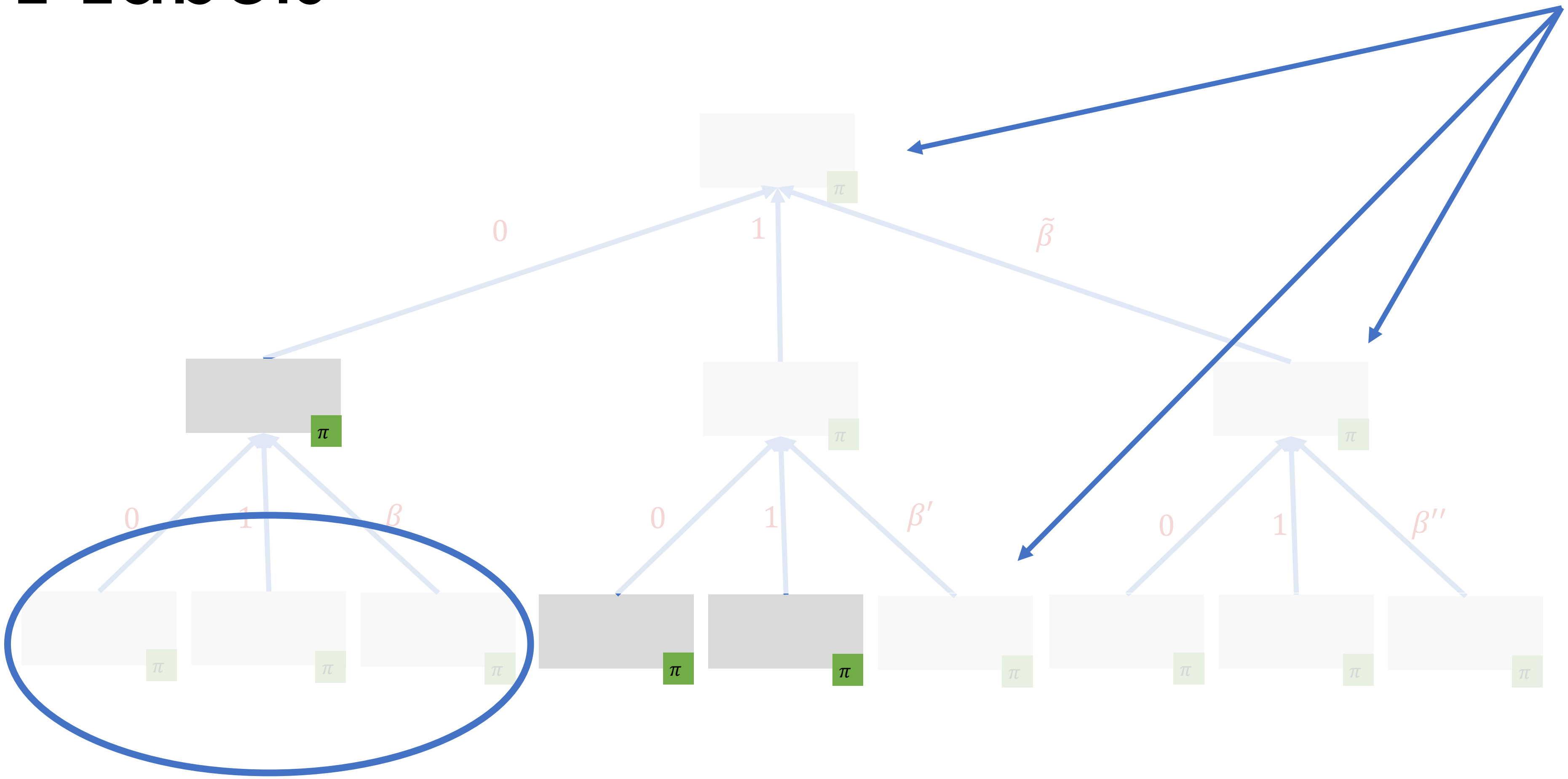
rSVL Labels



Merged and discarded

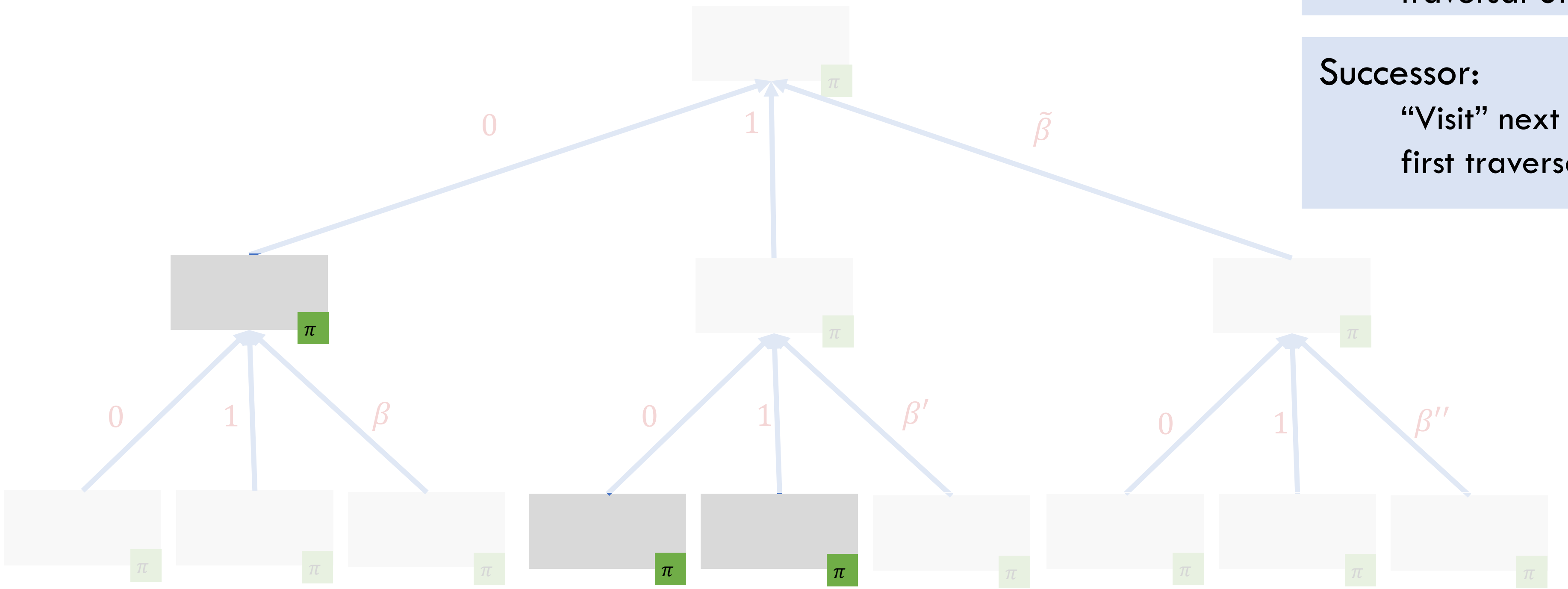
rSVL Labels

Haven’t reached yet



Merged and discarded

rSVL Labels



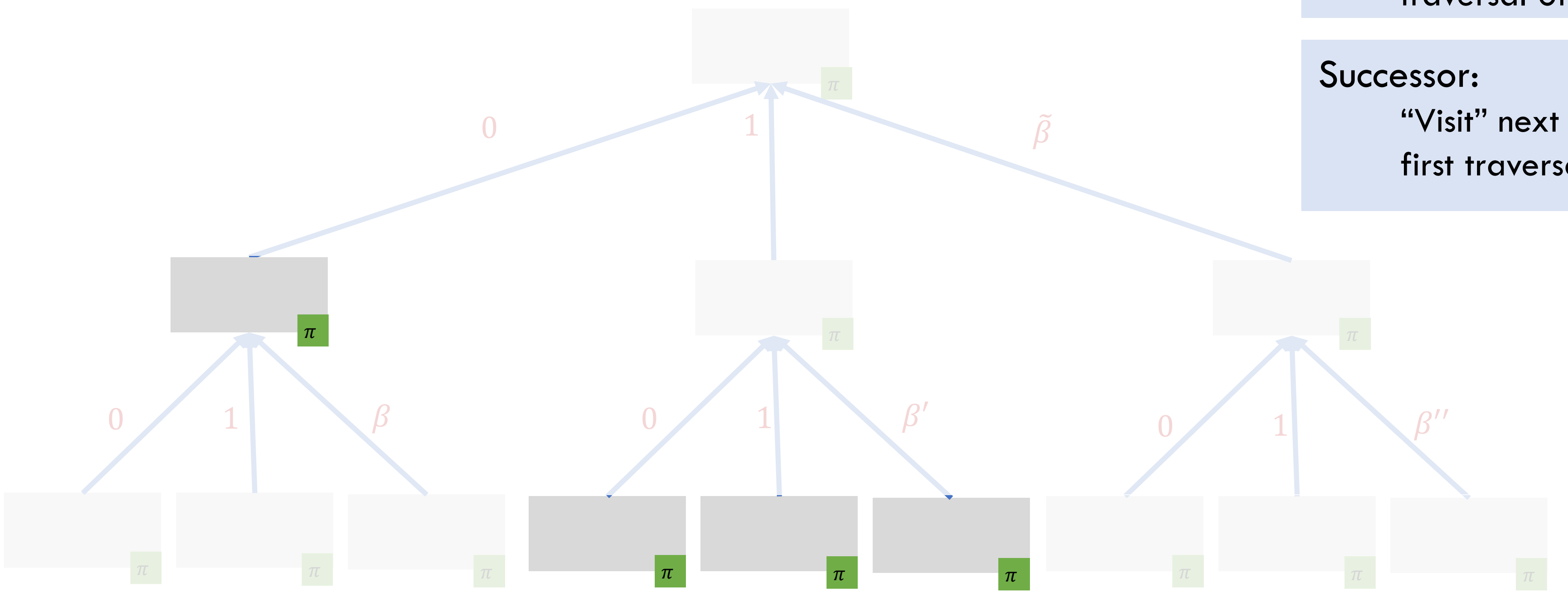
rSVL Labels:

Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:

"Visit" next node on the depth first traversal.

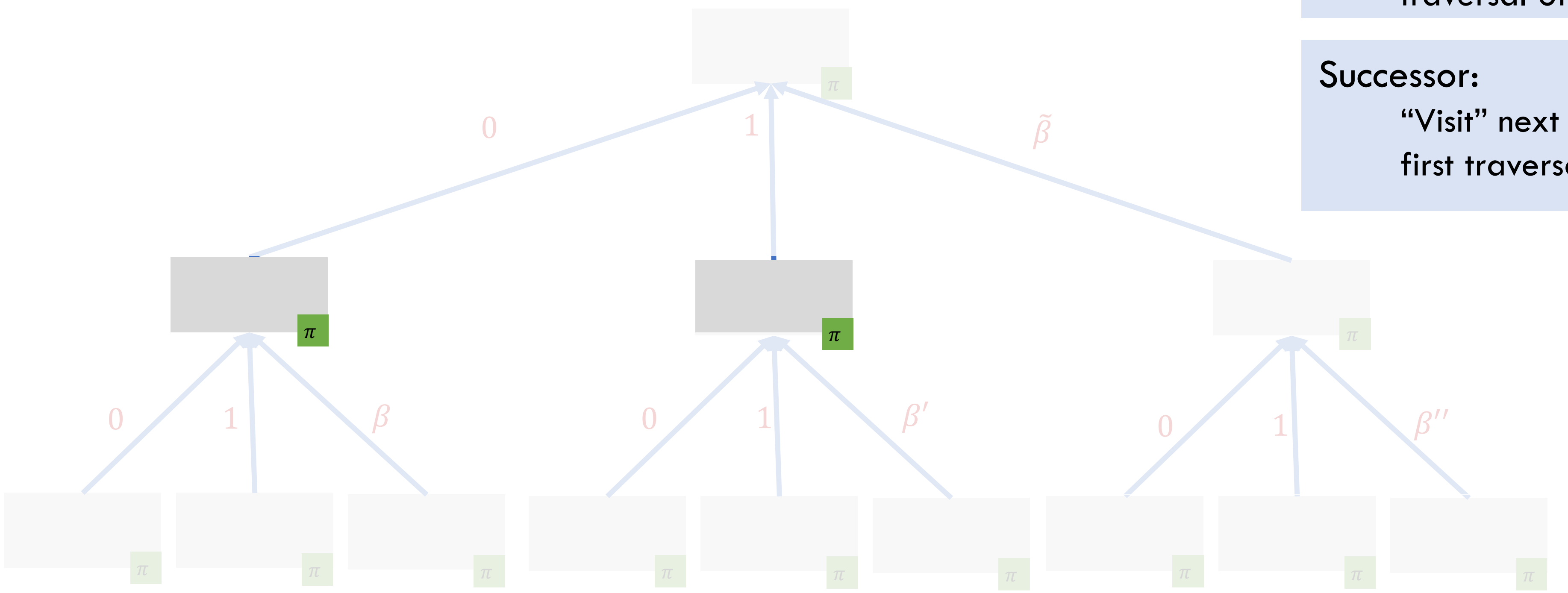
rSVL Labels



rSVL Labels:
Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:
"Visit" next node on the depth first traversal.

rSVL Labels



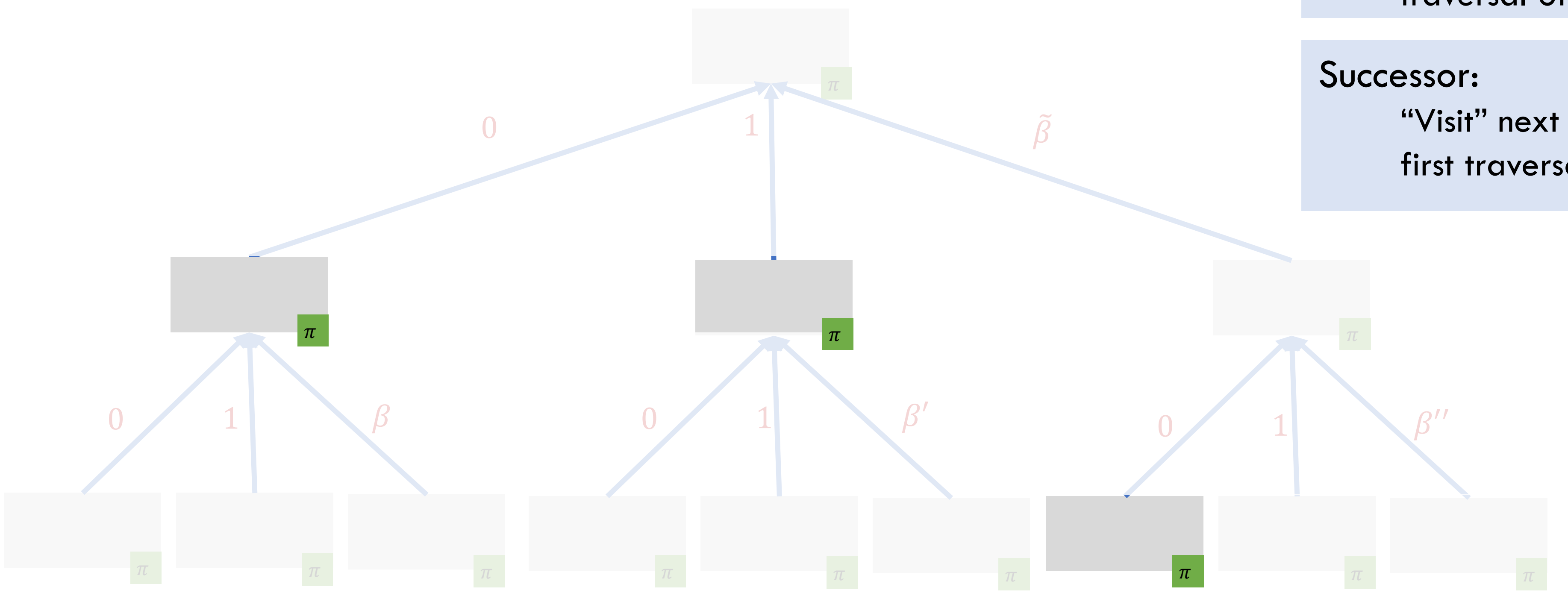
rSVL Labels:

Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:

“Visit” next node on the depth first traversal.

rSVL Labels



rSVL Labels:

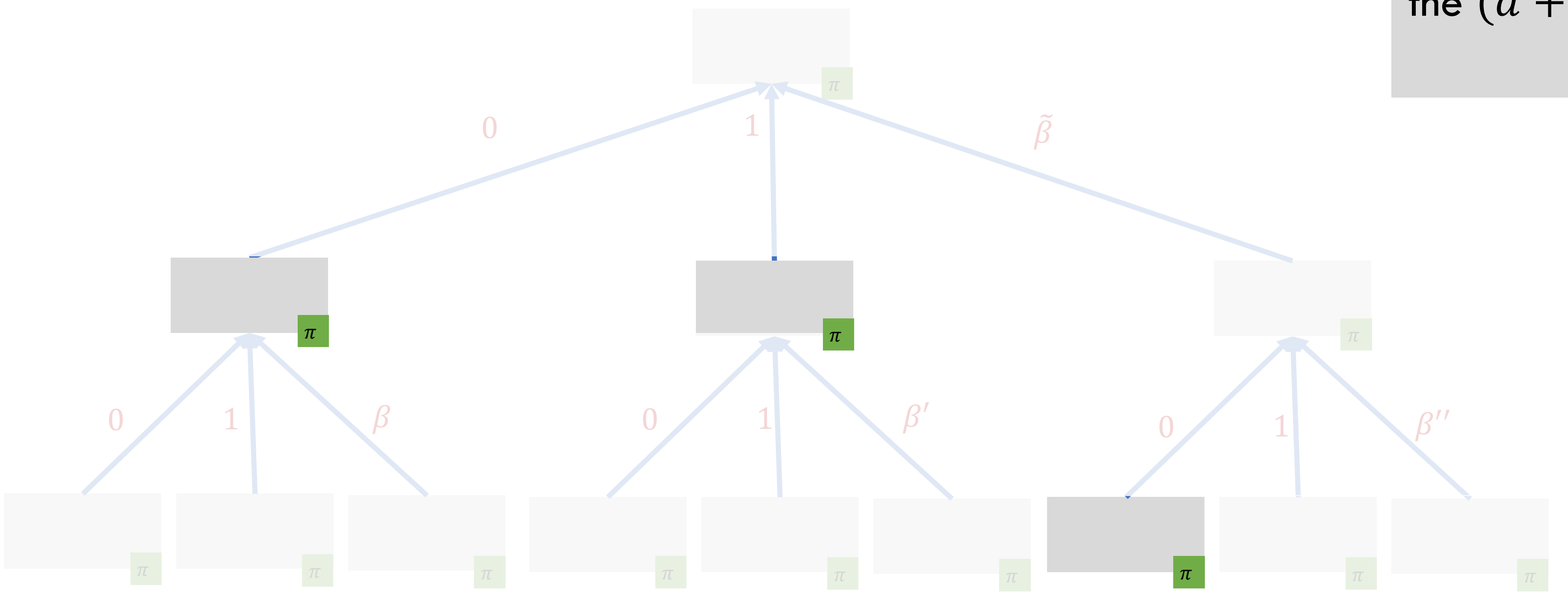
Nodes + Proofs on the boundary of a depth first traversal of tree.

Successor:

"Visit" next node on the depth first traversal.

rSVL Labels

Depth first traversal of the $(d + 1)$-ary tree



Verifying i-th state:

1. Determine which nodes are active in i-th step of depth first traversal.
2. Verify proofs in each active node.

Putting it together

Compute $\sum_{z\in\{0,1\}^n} p(z)$ in a continuous verifiable manner

Compute root of $(d+1)$-ary tree

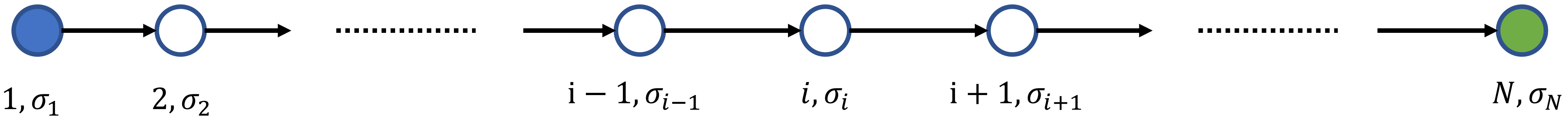
steps

$$P(N) = (d+2)P(N/2) + \text{poly}(n)$$

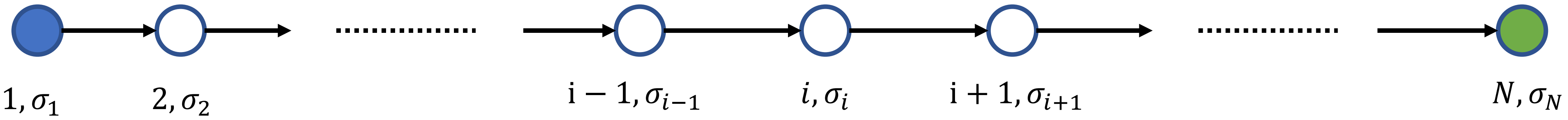
Proof size

$$S(N) = S(N/2) + \text{poly}(n)$$

Basic Idea: Long Computation + SNARGs

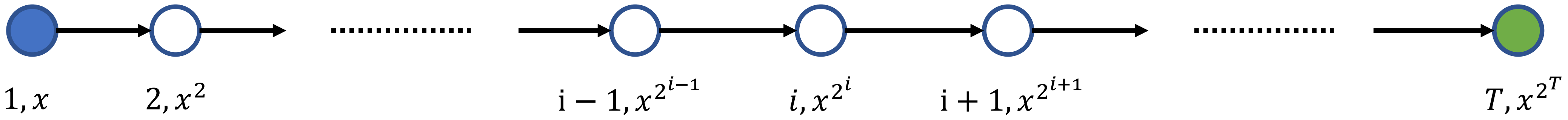


Basic Idea: Long Computation + SNARGs



Reduce Iterated Squaring to rSVL

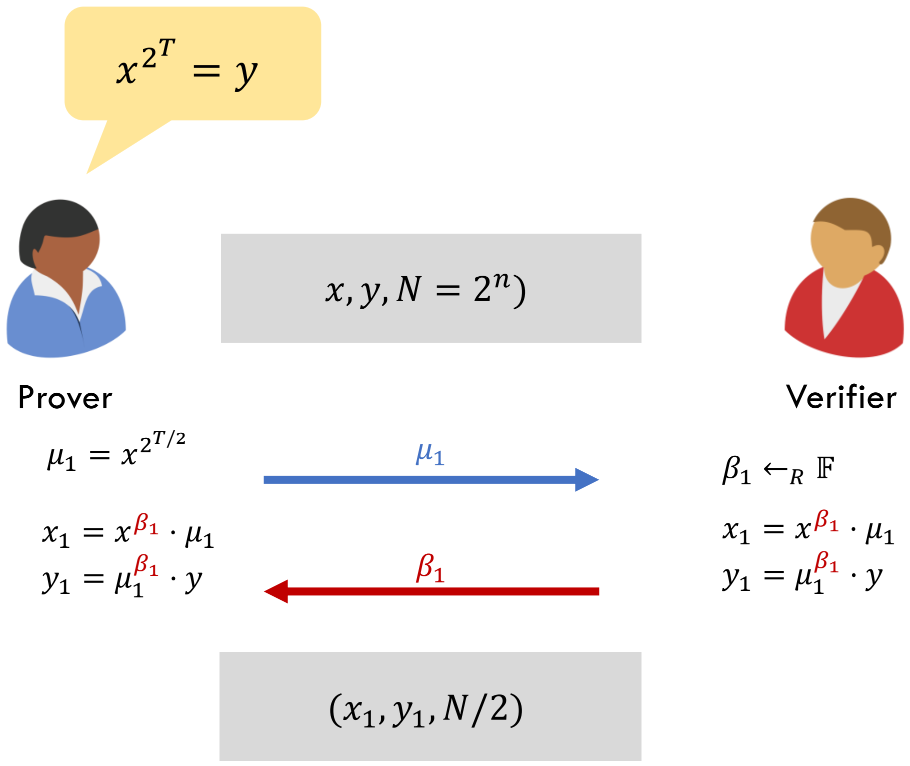
Basic Idea: Long Computation + SNARGs



Reduce Iterated Squaring to rSVL

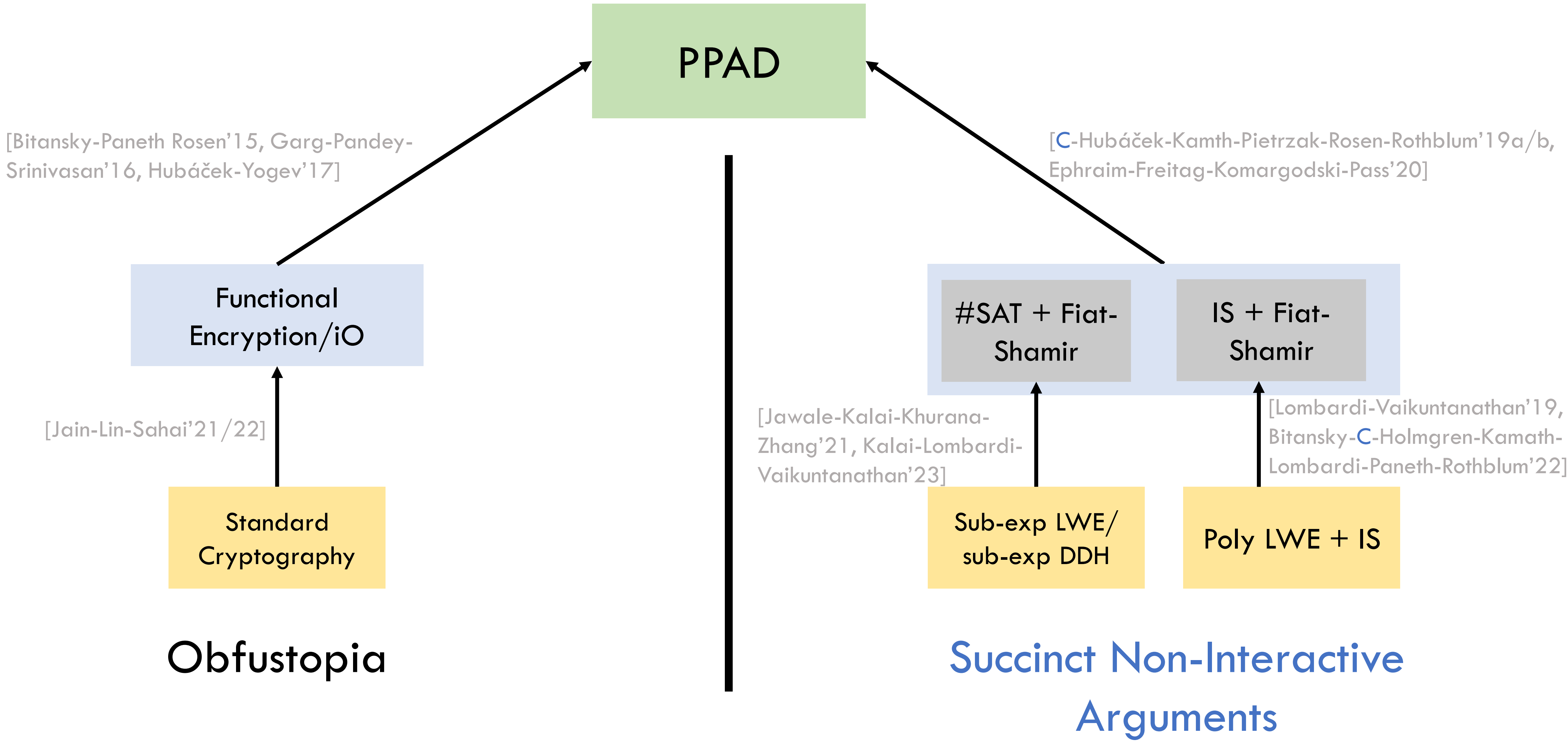
Outline and Batch for Iterated Squaring

[C-Hubáček-Kamth-Pietrzak-Rosen-Rothblum’19,Ephraim-Freitag-Komargodski-Pass’20]



[Pietrzak’19]

PPAD Hardness from Standard Cryptographic Assumptions



Open Problems

PPAD from poly LWE (proof of quantum hardness).

PPAD hardness without implying CLS hardness.

PPAD from Factoring.

Thank you. Questions?

Arka Rai Choudhuri

arkarai.choudhuri@ntt-research.com