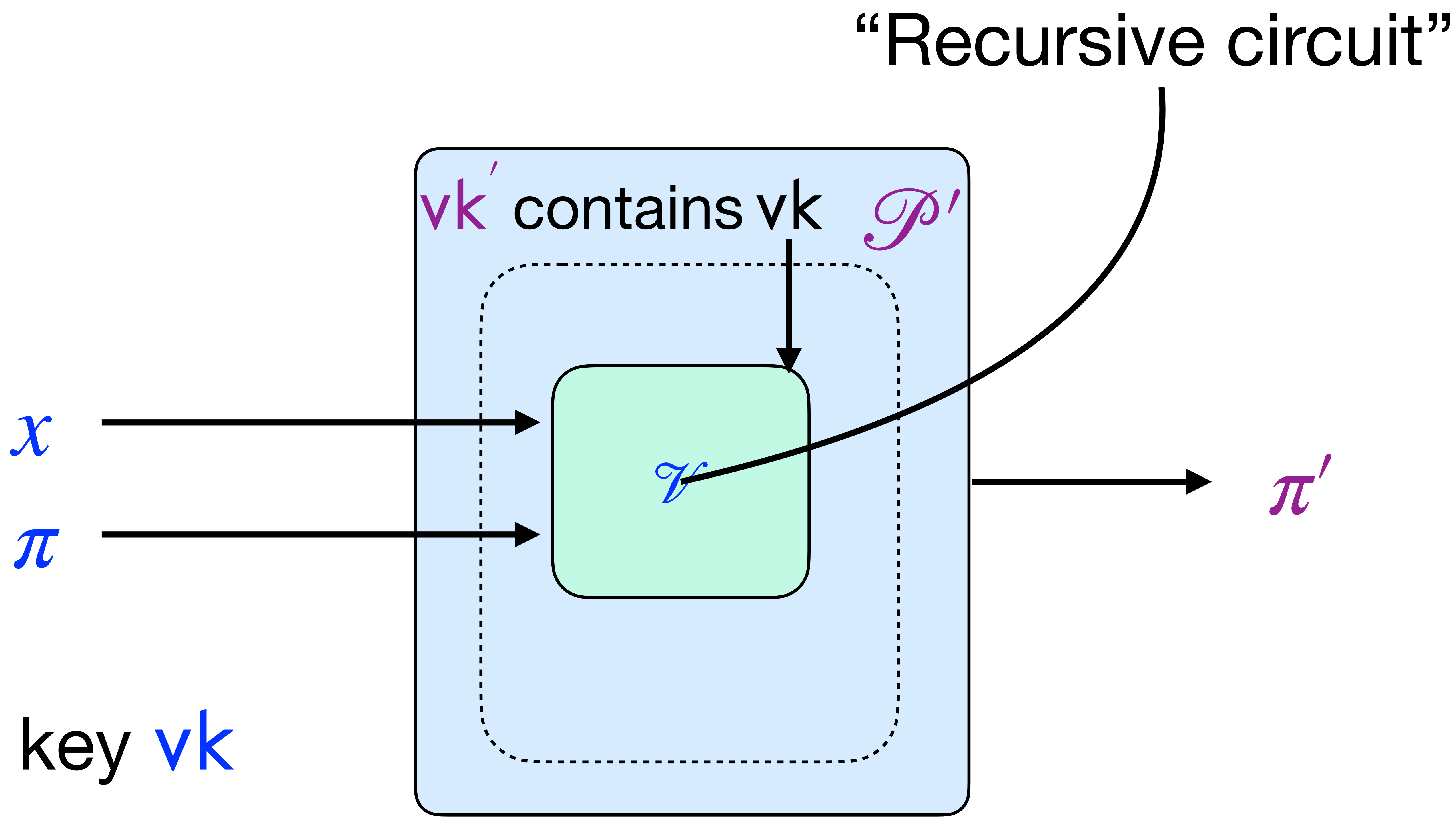
Recursive Proofs: Definitions, Applications, Security and Constructions

Benedikt Bünz (NYU)

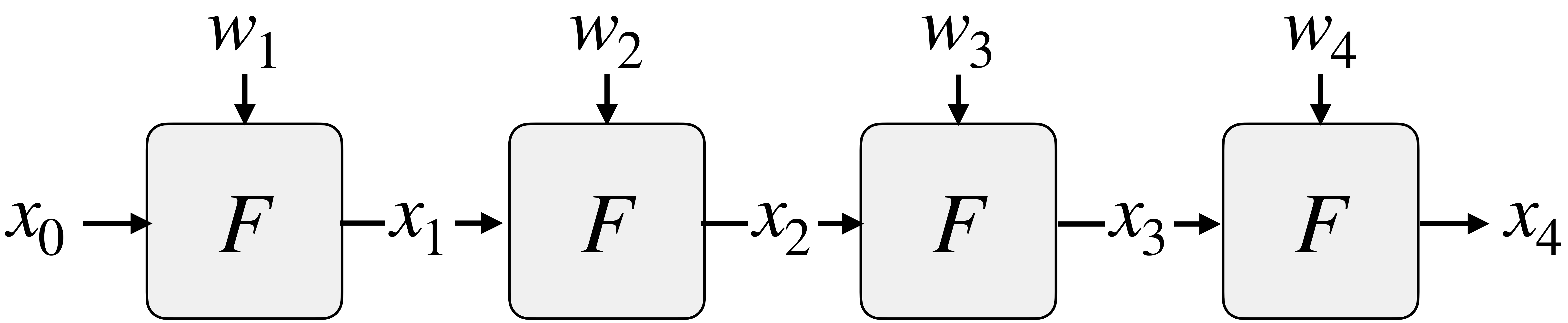
Recursive Proofs [Val08]

- Two SNARK systems $(\mathscr{P}, \mathscr{V})$, $(\mathscr{P}', \mathscr{V}')$
  - Sometimes they are the same
- $\mathscr{P}'$ proves that
  - it knows a proof π for a statement x
  - In a language indexed by a verification key vk
  - Such that $\mathscr{V}$ accepts π, for statement x and verification key vk
- Knowledge soundness of $(\mathscr{P}', \mathscr{V}')$ implies we can extract π



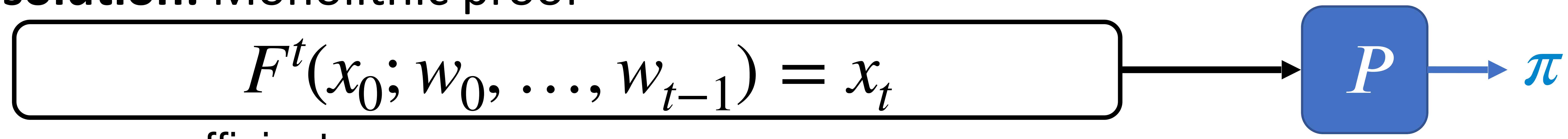
Motivation 1:

- **Goal:** Prove sequential computations



- "$x_t = F^t(x_0; w_1, \ldots, w_t)$ for some $w_1, \ldots, w_t$"

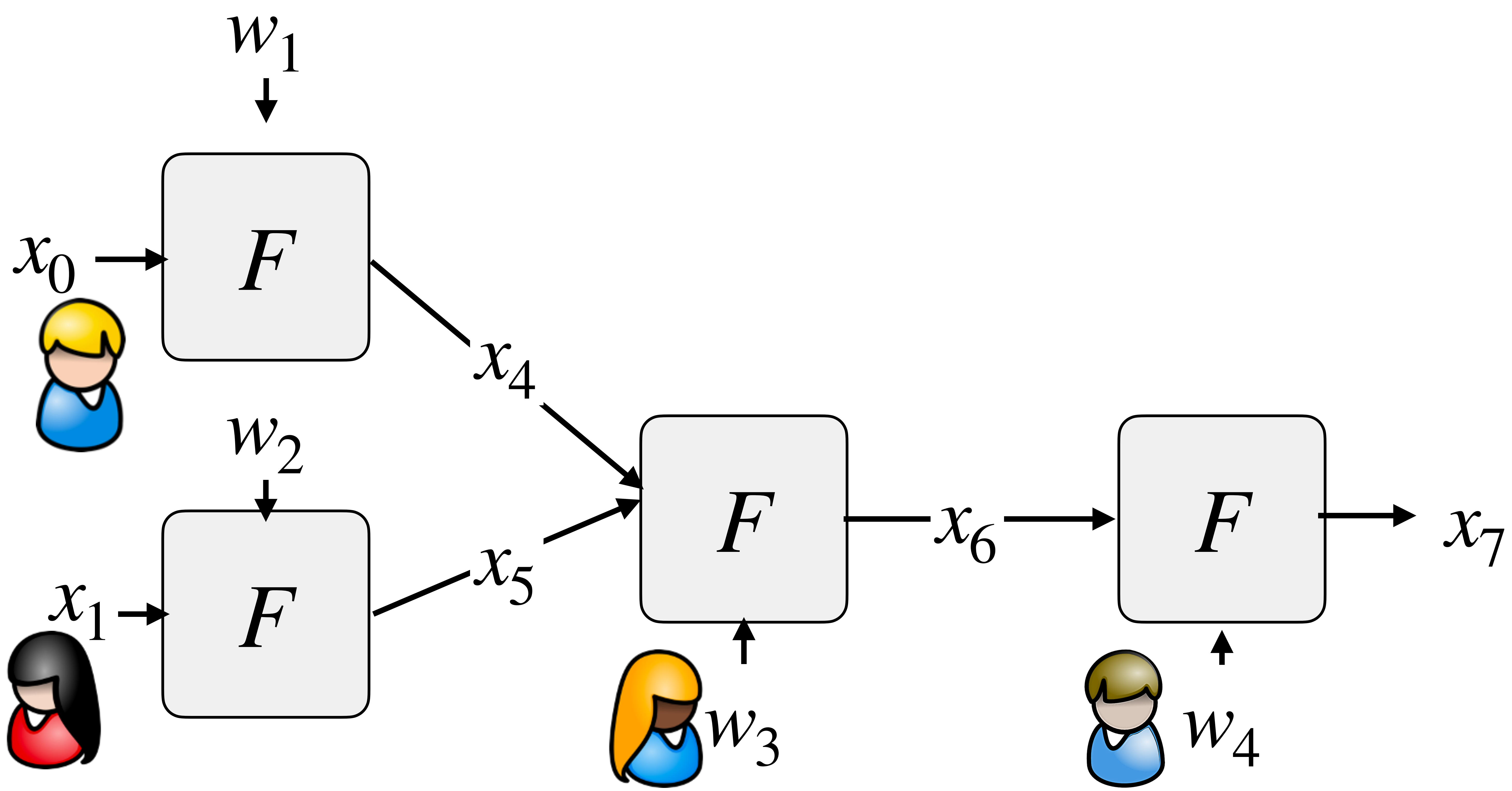
Naive solution: Monolithic proof



- Not memory-efficient
- Super-linear prover is super-linear in $t \cdot |F|$
- Additional steps requires reproving everything

Motivation 2

- **Goal:** Handing off computation

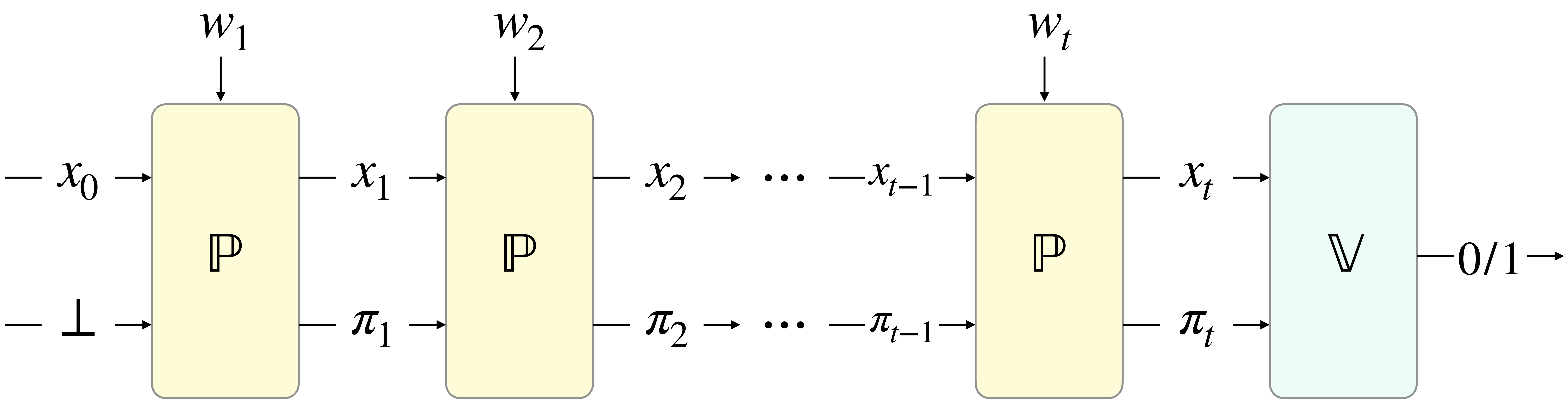


- Each party wants to verify the inputs. Some wants to check the entire computation

Naive solution: Each party creates a proof, attach all proofs.

- Linear in number of steps

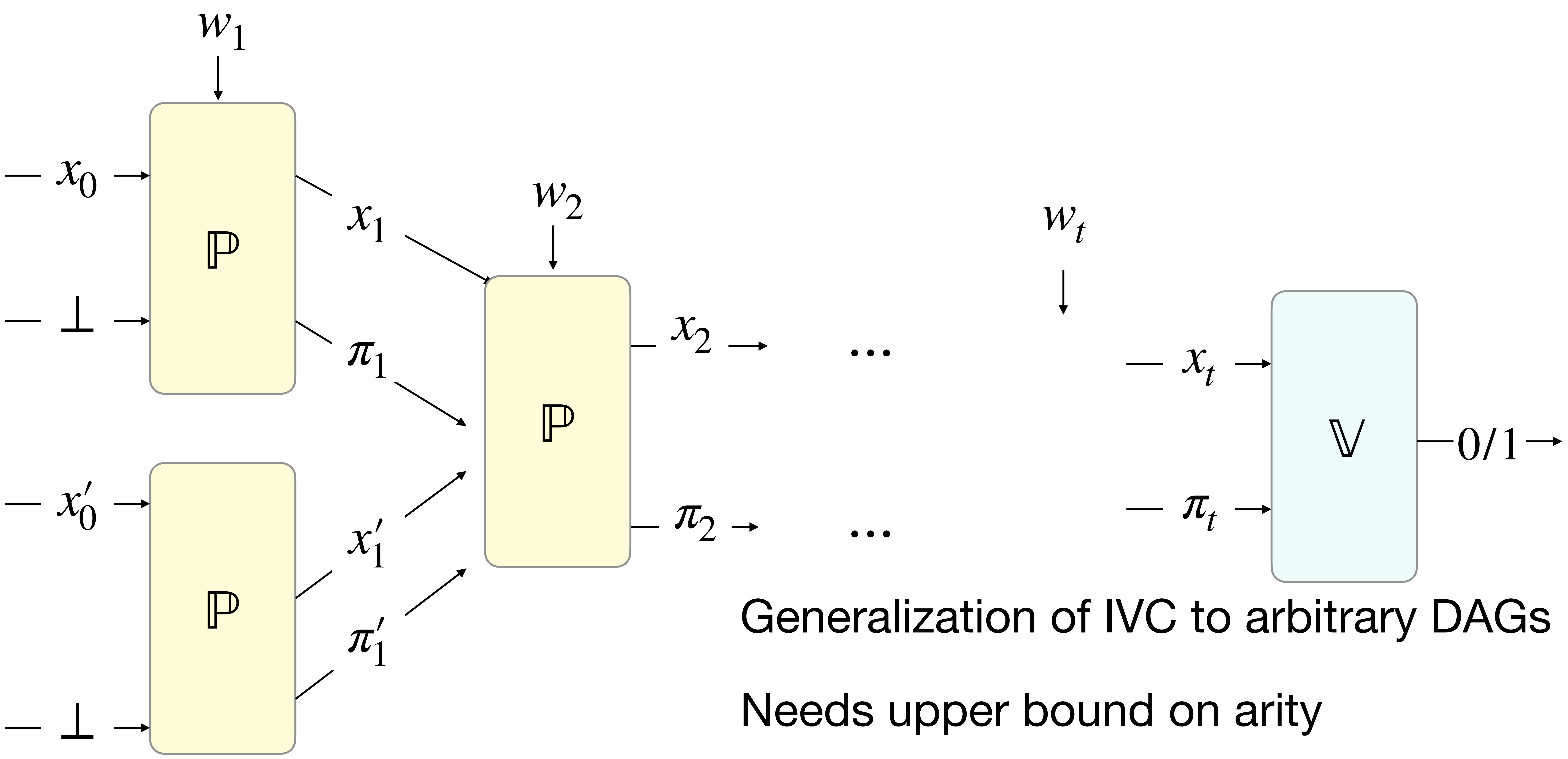
Incrementally verifiable computation [Val08]



- **Completeness:** Given valid proof π_{i-1} for x_{i-1}, $\mathbb{P}$ generates a valid proof π_i for $x_i := F(x_{i-1}, w_i)$
- **Knowledge soundness:** Given valid proof π_t for x_t, extract witnesses $w_1, \ldots, w_t$ such that $x_t = F^t(x_0; w_1, \ldots, w_t)$
- **Efficiency:** Proof size and prover/verifier runtime should be independent of t

IVC for P can be build from batch arguments[KPY19,DGKV22, PP23] (we will focus on NP)

Proof Carrying Data (PCD) [CT10,BCCT13]



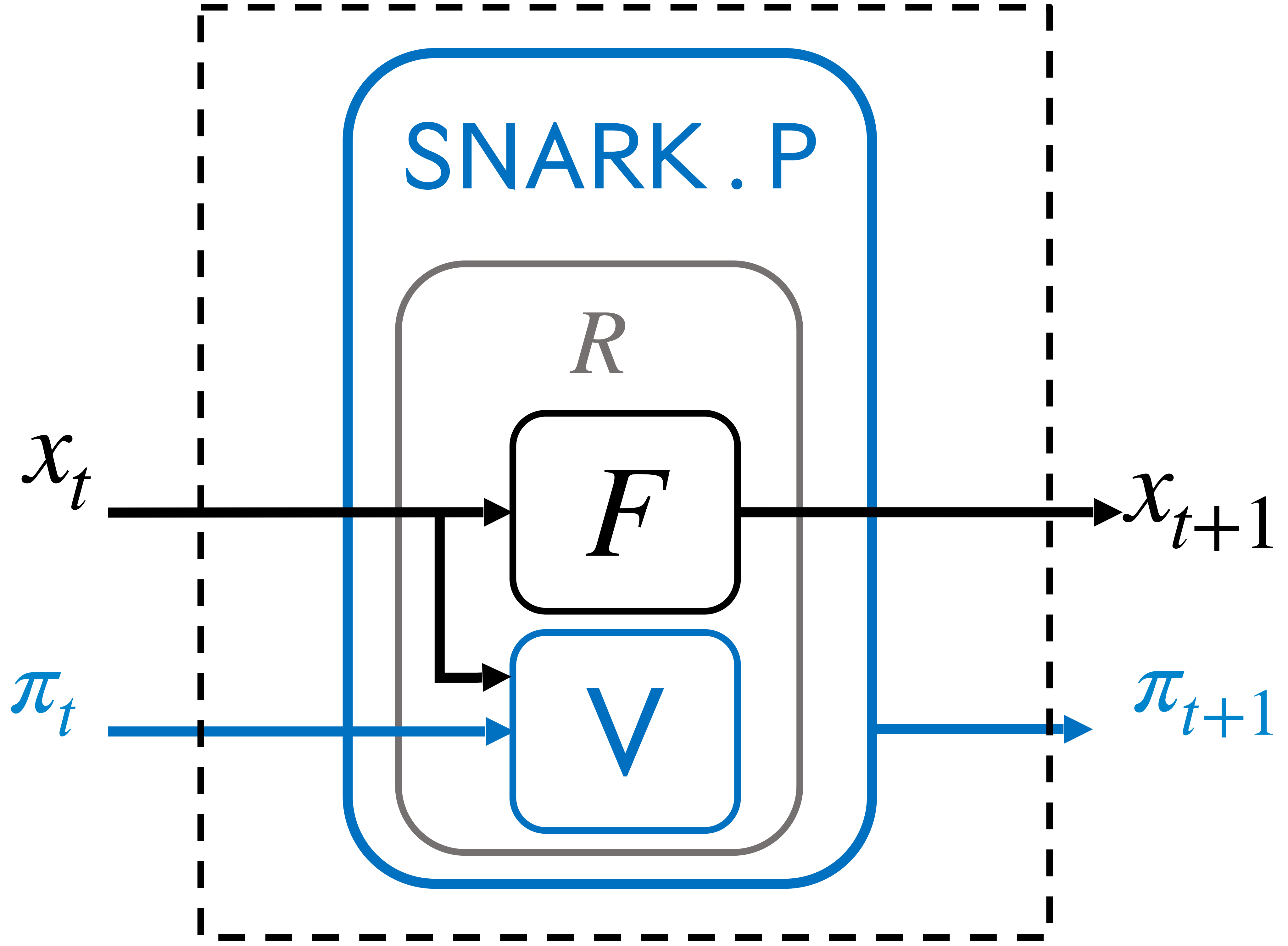
Generalization of IVC to arbitrary DAGs

Needs upper bound on arity

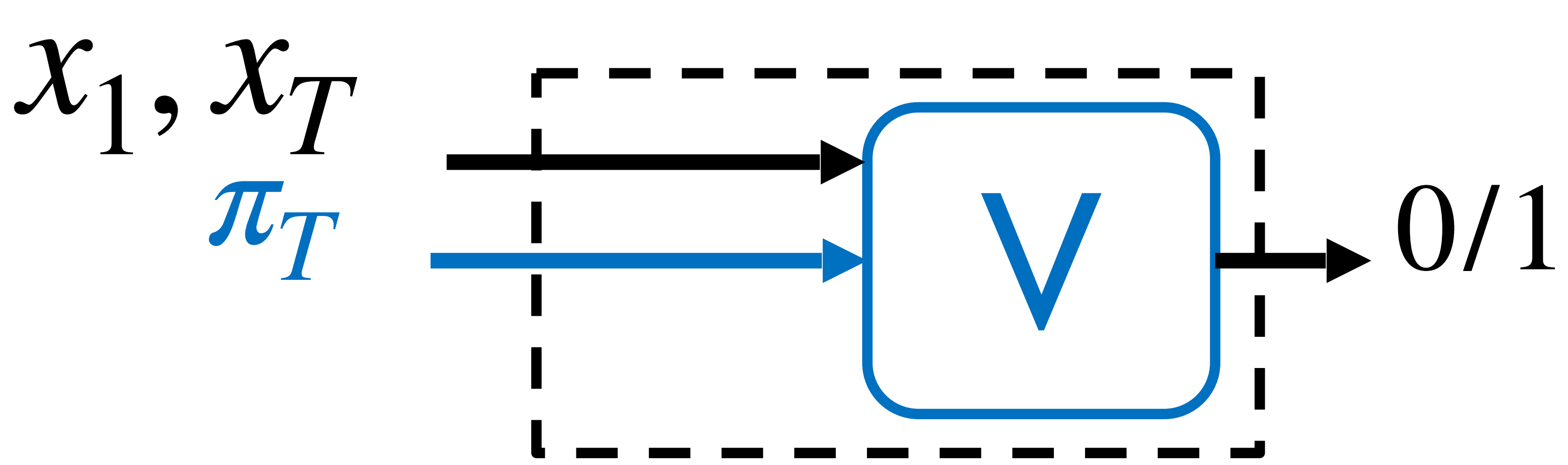
IVC from recursive composition of SNARKs

[BCCT13, COS20]

$\mathrm{IVC}.\mathbb{P}_F$



$\mathrm{IVC}.\mathbb{V}_F$



Completeness:

Follows from SNARK completeness

Soundness:

Recursively extract transcript using SNARK knowledge soundness [BCCT13, COS20]

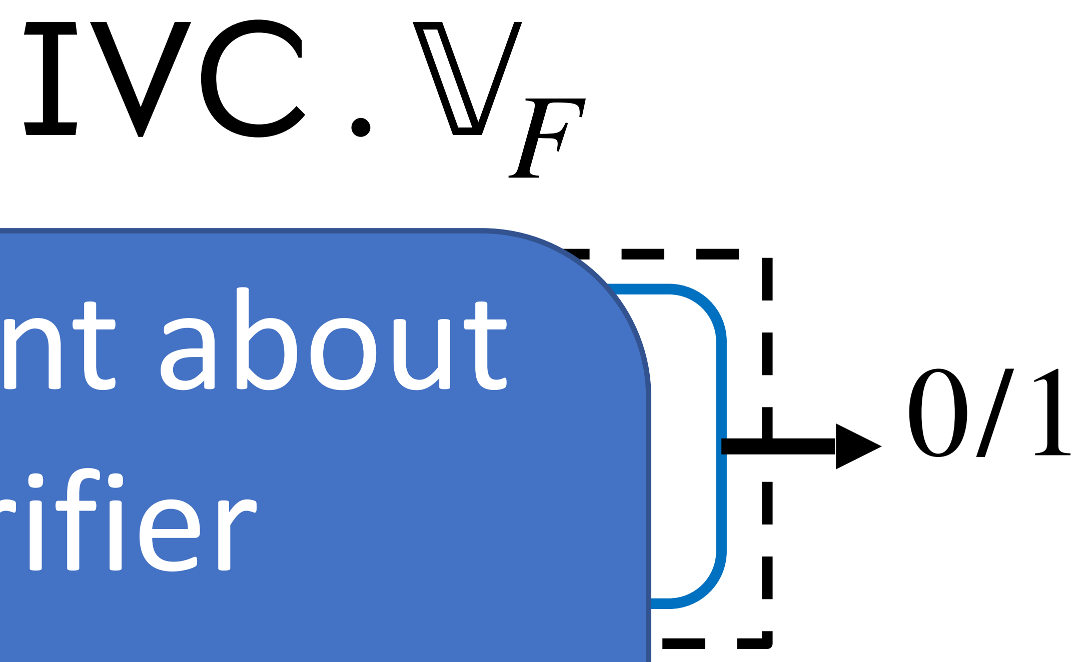
Multiple issues (we'll discuss in a minute)

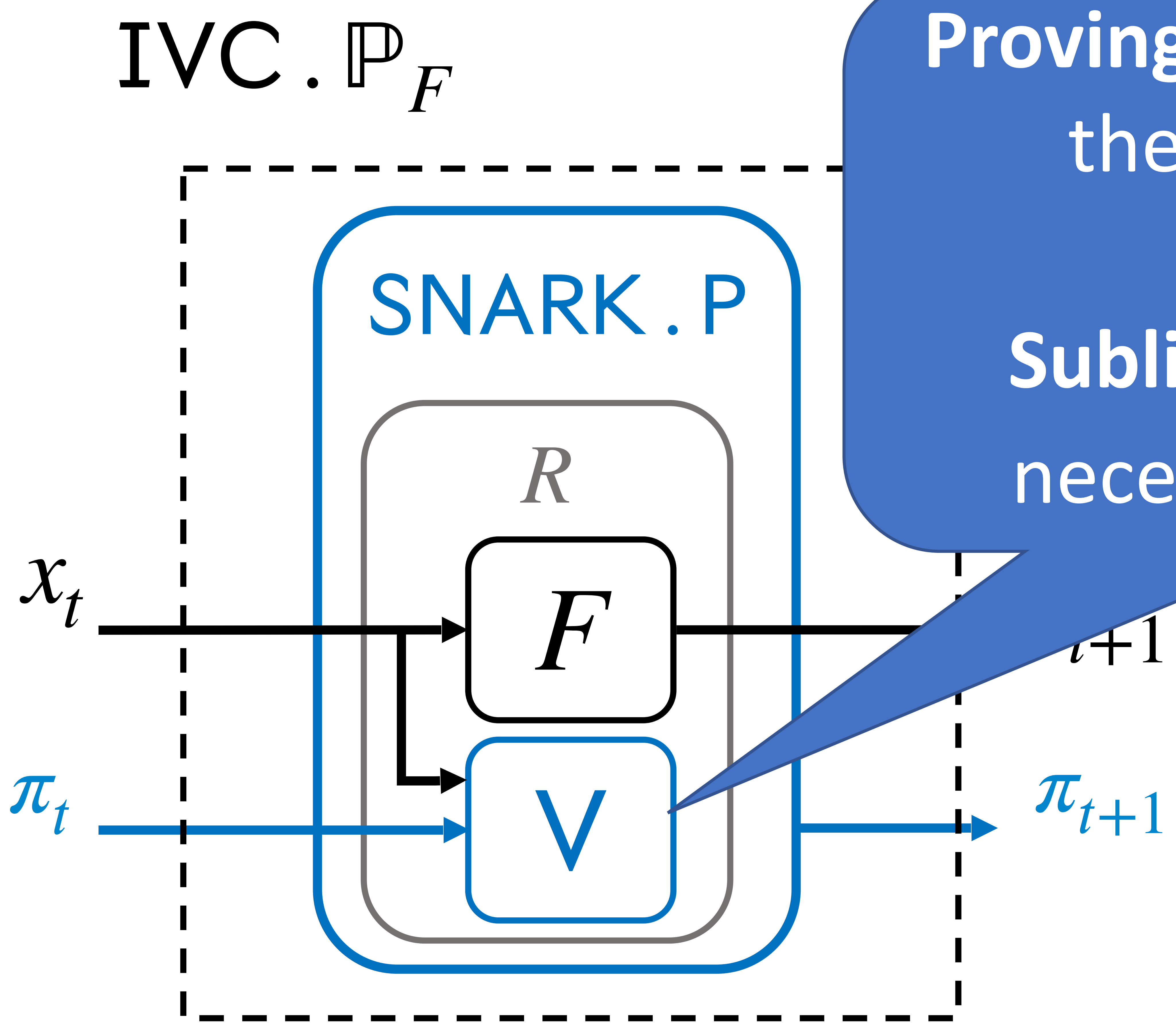
Efficiency:

$|\pi| =$ size of a SNARK proof for R...

IVC from recursive composition of SNARKs

[BCCT13, COS20]





Proving a statement about the SNARK verifier

$\Downarrow$

Sublinear verification necessary to bound R

...npleteness

...script using SNARK

knowledge soundness [BCCT13, COS20]

Multiple issues (we'll discuss in a minute)

Efficiency:

$|\pi| =$ size of a SNARK proof for R...

Application 3: Property preserving SNARKs

- **Goal:** Improving SNARK prover properties

SNARK A → SNARK A'

- Fast sequential Prover
- Non parallel
- Large memory
- Large CRS
- "Large" verifier

- Fast parallel Prover
- Constant memory
- Constant CRS
- Constant size verifier

Solution: Break up function F into T uniform steps F' of size $\frac{|F|}{T}$.

- Build binary PCD tree of depth $\log(T)$ for predicate F'.
- **T** parallelism, memory, CRS and Verifier for **F' +** Circuit(V_A)

Application 4: SNARK composition

- **Goal:** Combining SNARKs with different tradeoffs

SNARK A + SNARK B = SNARK C

SNARK A	SNARK B	SNARK C
• Fast Prover • “Slow” Verifier • “Large” Proofs	• Slow Prover • Fast Verifier • Small Proofs • Zero-Knowledge	• Fast Prover • Fast Verifier • Small Proofs • Zero-Knowledge

Solution: Use SNARK B to prove correctness of SNARK A

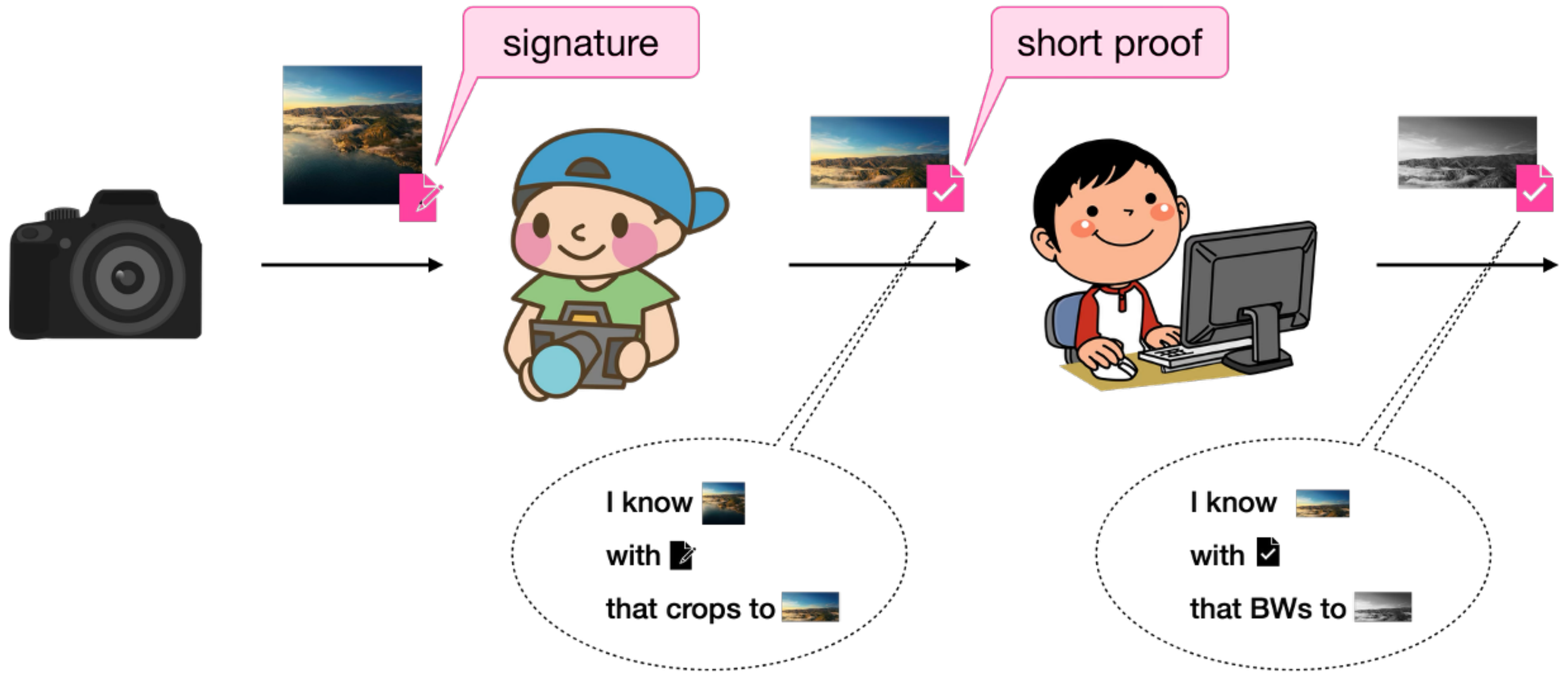
Prover runtime: P_A on $|F|$ + P_B on Circuit(V_A)

Verifier: V_B, Proof size: π_B

Many more applications

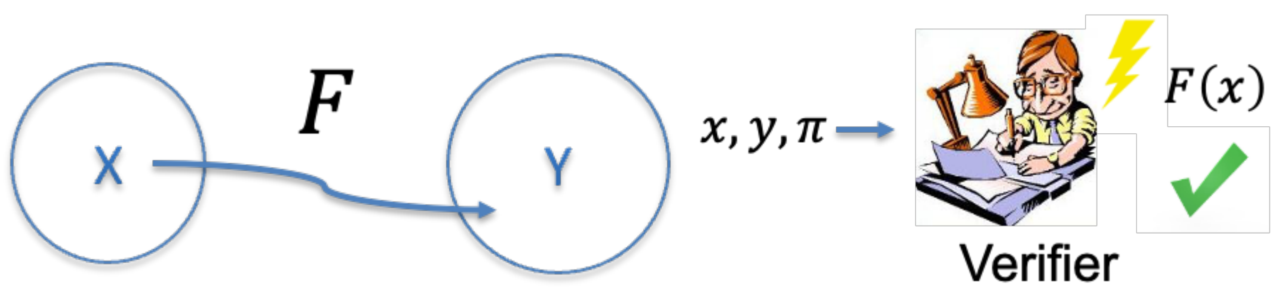
Image provenance

[NT16]



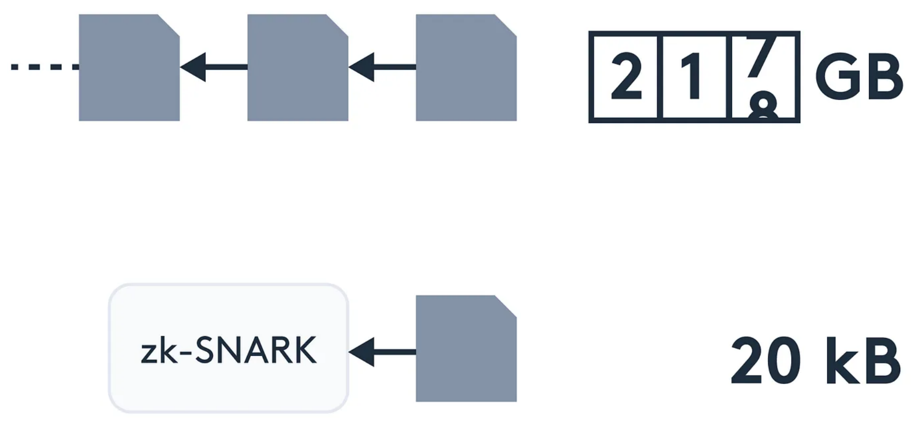
Verifiable Delay functions

[BBBF20]



Succinct Blockchains

[BMRS20,KB20,CCDW20]



- Byzantine agreement [BCG20]
- ZK cluster computing [CTV15]
- Enforcing language semantics across trust boundaries [CTV13]
- Private smart contracts[BCCGMW18]
- Signature aggregation [KZHB25]
- …

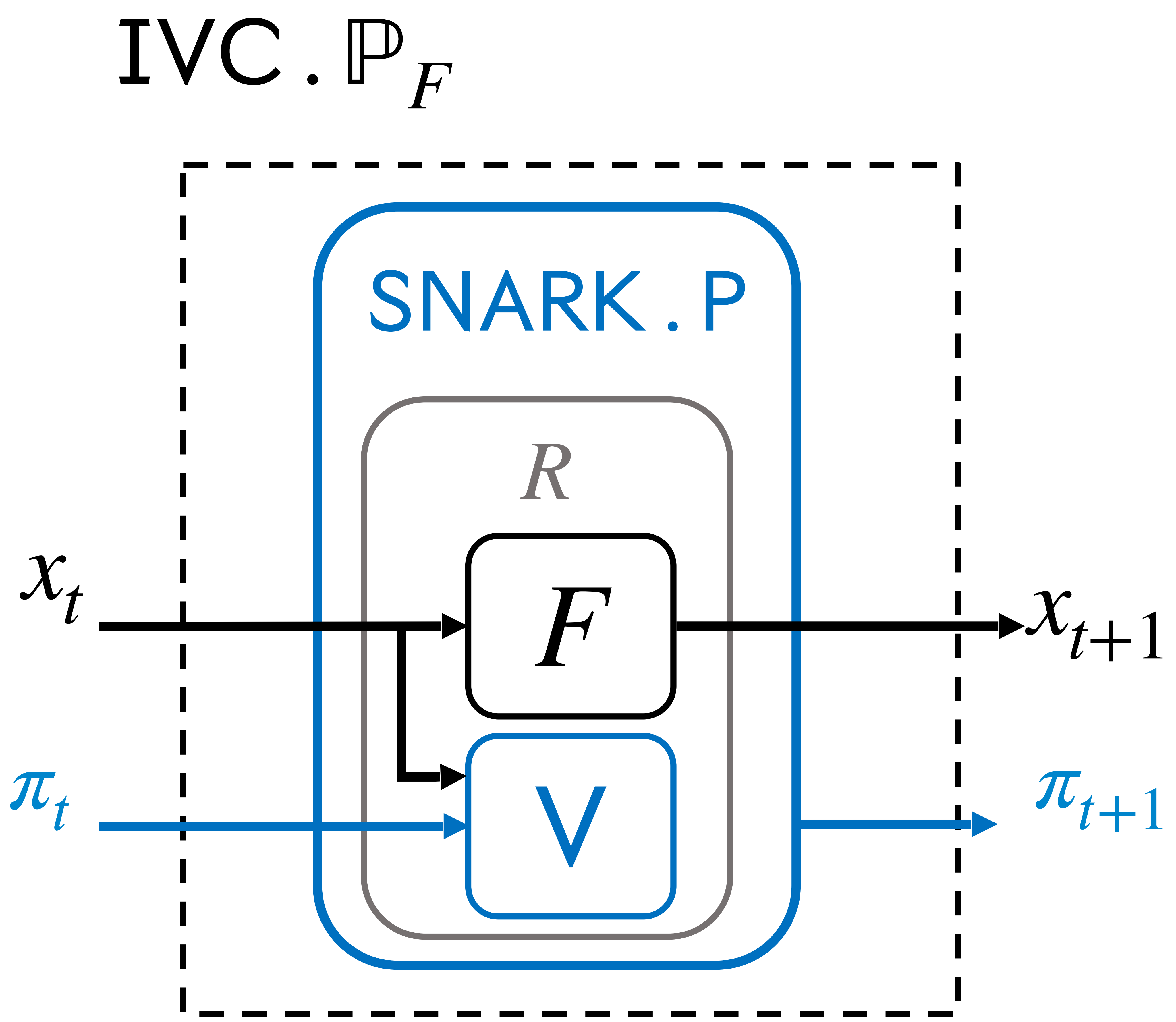
Real world deployments (AI-SC's darling)

- Recursive proofs are widely deployed!

- Vital to understand their security and improve constructions!

Security analysis and problems

Security issues: Arithmetizing V



- R contains V $\implies$ V can't contain oracles
- We need to implement V as a circuit

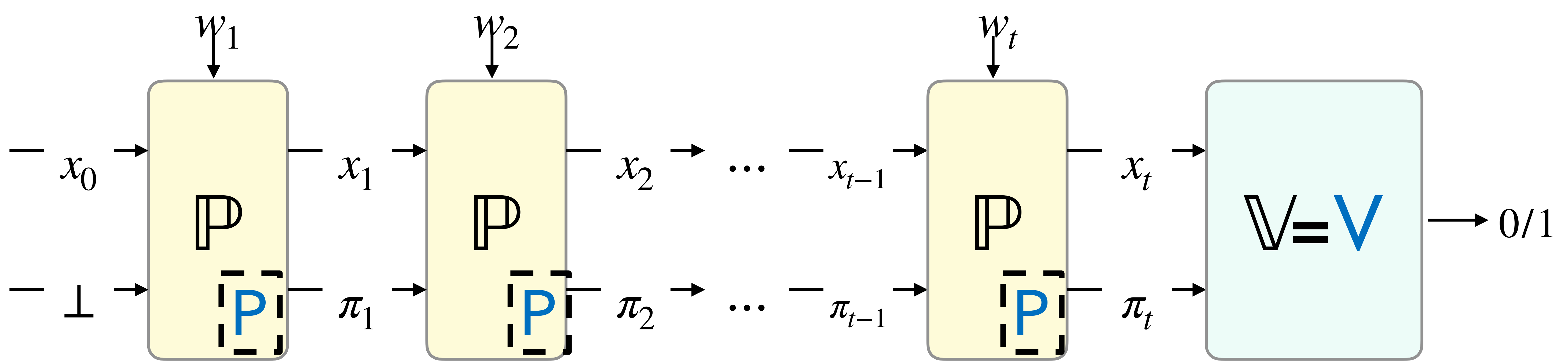
- ***Security jump***: $(\mathsf{P}^\rho, \mathsf{V}^\rho)$ *secure in the RO implies that* (V, P)=*Fiat-Shamir*$(\mathsf{P}^\rho, \mathsf{V}^\rho)$ *is secure in the standard (CRS) model.*

- Generically not true[CGH98,Bar01,GK03]
- Recent attack on GKR[KRS25]
  - Attack relies on evaluating FS-Hash inside proof system
  - Recursion relies on this ability

Security issues: Arithmetizing V

- Attempt 1: Build SNARK in the RO that proves statement about the RO?
  - Impossible [BCG24]
- Attempt 2: Extend RO model to enable end-to-end analysis of PCD
  - Early attempts required secure hardware [CT10,CCS22]
  - Arithmetized Random Oracle Model [CCGOS23] augments the random oracle with an additional arithmetization oracle. Heuristically, the RO is replaced with SHA256, and the arithmetization with a circuit of SHA256.
  - AROM suffices to build PCD
  - But FS-attacks are not captured by the AROM (The insecure SNARK is still secure)
- Open problem: Build model that is sufficient to capture attacks but enables end-to-end PCD construction (candidates [Zha22,AY25])

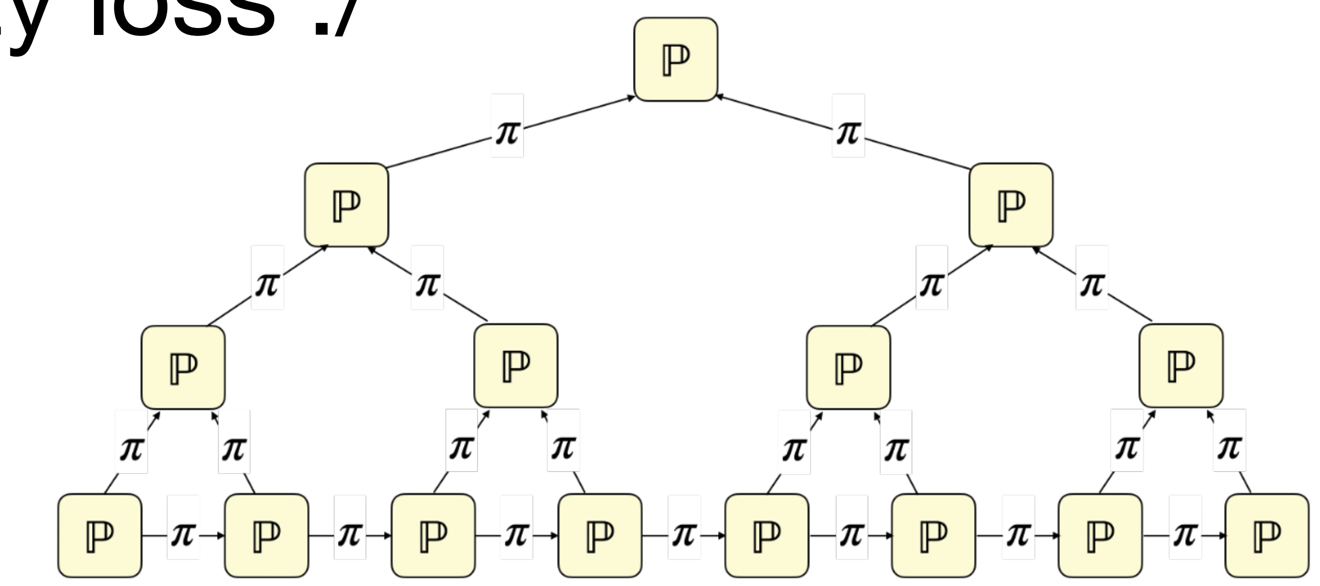
Security Issues: Extraction



- IVC extractor calls the SNARK extractor using (π_t, x_t) to extract $\pi_{t-1}, x_{t-1}, w_{t-1}$
- To extract from an internal SNARK (e.g. for step 2) we need to simulate a prover $\tilde{\mathsf{P}}$ for that SNARK.
- **Idea**: $\tilde{\mathsf{P}}$'s proofs are generated by invoking the extractor for the outer SNARKs
- **Problem**: each extractor can invoke each $\tilde{\mathsf{P}}$ up to poly(λ) times
- Thus the runtime of the extractor is poly$(\lambda)^{\text{depth}}$ $\implies$ depth must be **constant**

Security Issues: Extraction

- Constant-depth IVC/PCD only and with major security loss :/
- Old solution: **Decrease depth** **[BCCT13]**
  - Use tree-based IVC
  - With λ arity and constant depth we can support $\mathsf{poly}(\lambda)$ IVC steps
  - Still high security loss



- Practitioner's solution:
  - **Don't do anything**
  - Assume $\epsilon_{\mathsf{IVC}} \approx \epsilon_{\mathsf{SNARK}}$ (Soundness error of IVC is independent of depth)
  - No matching attack

Saving grace: Straightline extraction

- Assume the SNARK has a straight line (deterministic, one-shot) extractor
  - Then we don't get the exponential blowup (each extractor is called once)
  - Union bound: $\text{depth} \cdot \epsilon_{\mathsf{SNARK}}$ [CT10,CCGOS23]
  - Recently improved to $\epsilon_{\mathsf{PCD}} \approx \epsilon_{\mathsf{SNARK}}$ [CGSY23]
- **Problem 1:** Only able to construct straight-line extraction in idealized models
  - Heuristic assumption: straightline extraction in idealized model
  - $\implies$ straightline extractor for real-wold instantiation
- **Problem 2**: Some SNARKs of interest don't have straightline extractors
  - Example: SNARKs from non efficiently decodable codes.
    - Recent progress [RT24,BCFW25]
  - Straightline extraction (in ideal model) should become the norm for SNARKs

Open security problems

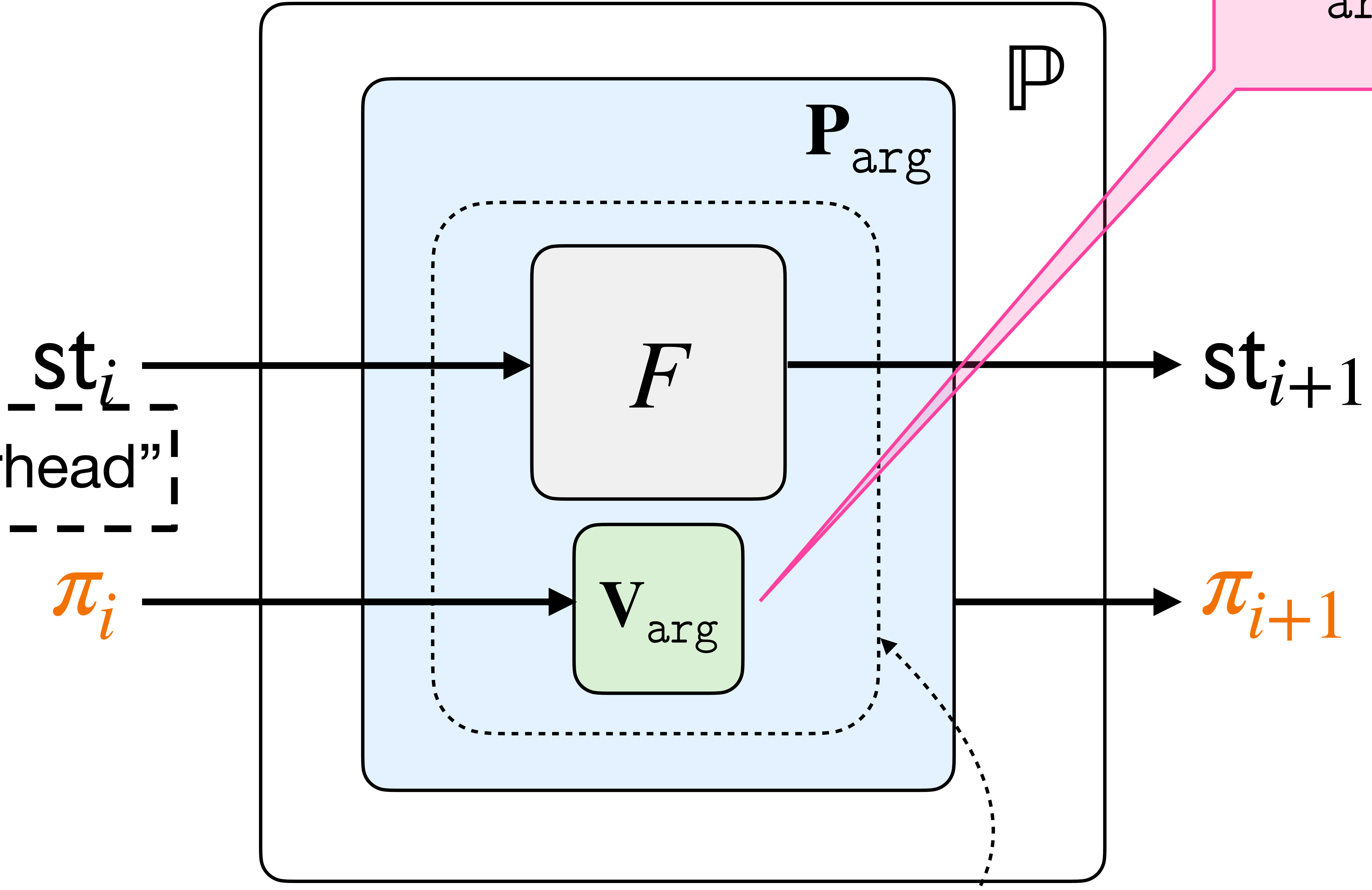
- Build IVC/PCD in standard model (see Surya’s talk on Thursday)
- Build a model that captures Fiat-Shamir attacks but enables proving security of known SNARK/PCD constructions
- Attacks against high-depth IVC/PCD (even contrived)
- “Straightline extraction” in standard CRS model (or similar condition)
- Proving straightline extraction for more protocols

Efficiency (concerns)

IVC from succinct arguments

concrete issues:
$\mathbf{P}_{\texttt{arg}}$ is slow
$\mathbf{V}_{\texttt{arg}}$ is large

recursive proof composition:
$\mathbf{P}_{\texttt{arg}}$ proves $\mathbf{V}_{\texttt{arg}}$ as a circuit



$|\text{circuit}(\mathbf{V}_{\texttt{arg}})|$ is "recursive overhead"

"there exists st_i with a valid proof π_i such that $\mathsf{st}_{i+1} = F(\mathsf{st}_i)$"

Recursive overhead is a bottleneck

- **Goal:** Improving SNARK prover properties

SNARK A → SNARK A’

- Fast sequential Prover
- Non parallel
- Large memory
- Large CRS
- Large verifier

- Fast parallel Prover
- Constant memory
- Constant CRS
- Constant size verifier

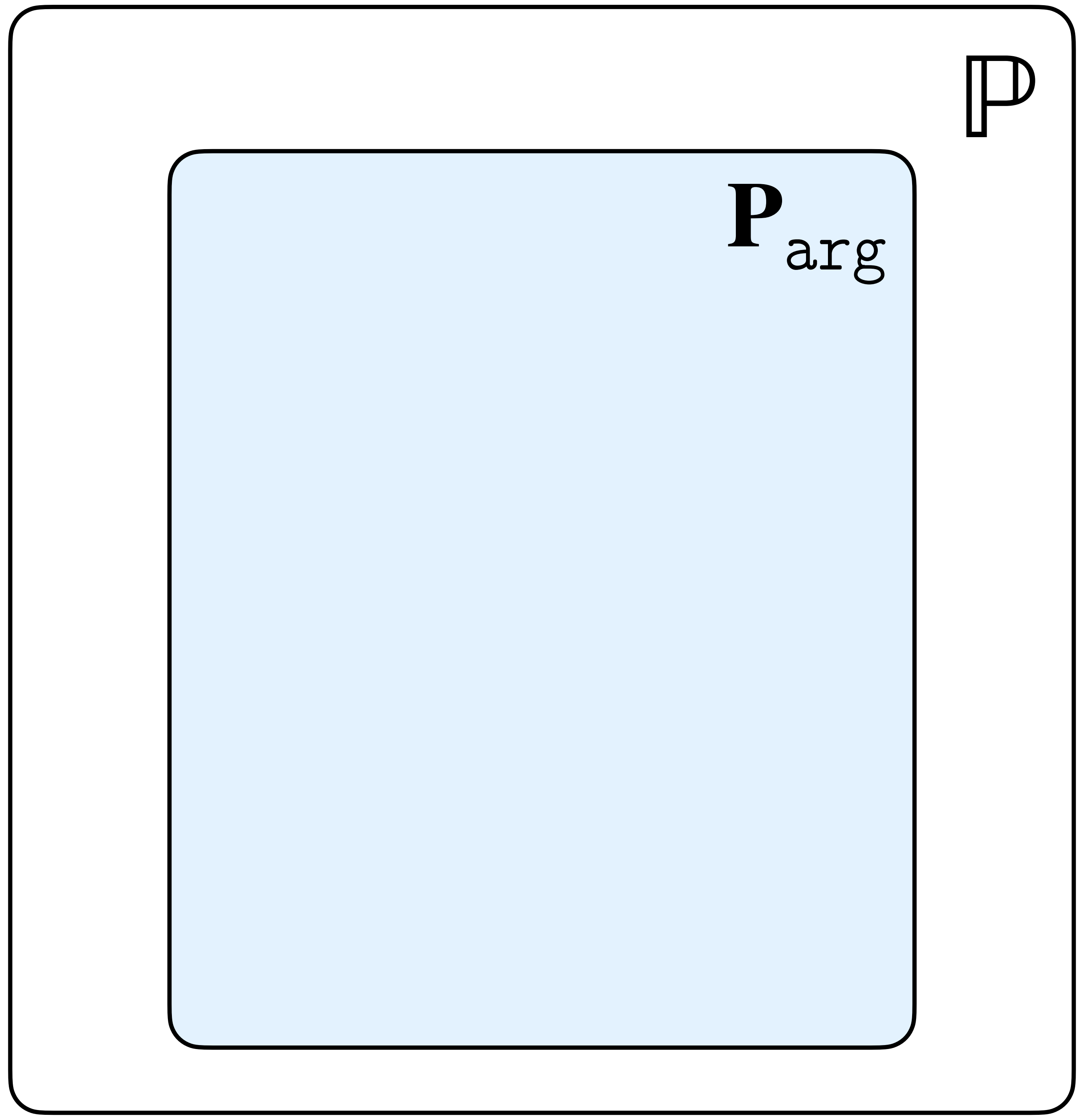
Smaller Circuit
$\Downarrow$
Larger T

Solution: Break up function F into T uniform steps F’ of size $\frac{|F|}{T}$.

- Build binary PCD tree of depth $\log(T)$ for predicate F’.
- **T** parallelism
- Memory, CRS and Verifier for **F’** + Circuit(V_A)

SNARK prover is a bottleneck

- PCD prover runs SNARK prover
- SNARK provers have large constants
- Many SNARKs have strong assumptions
  - E.g. SNARKs in DLOG groups
- Most efficient SNARKs have large proofs
  - Linear-time SNARKs have MB sized proofs
  - Leads to large recursive overheads



Are SNARKs necessary to build IVC/PCD*?

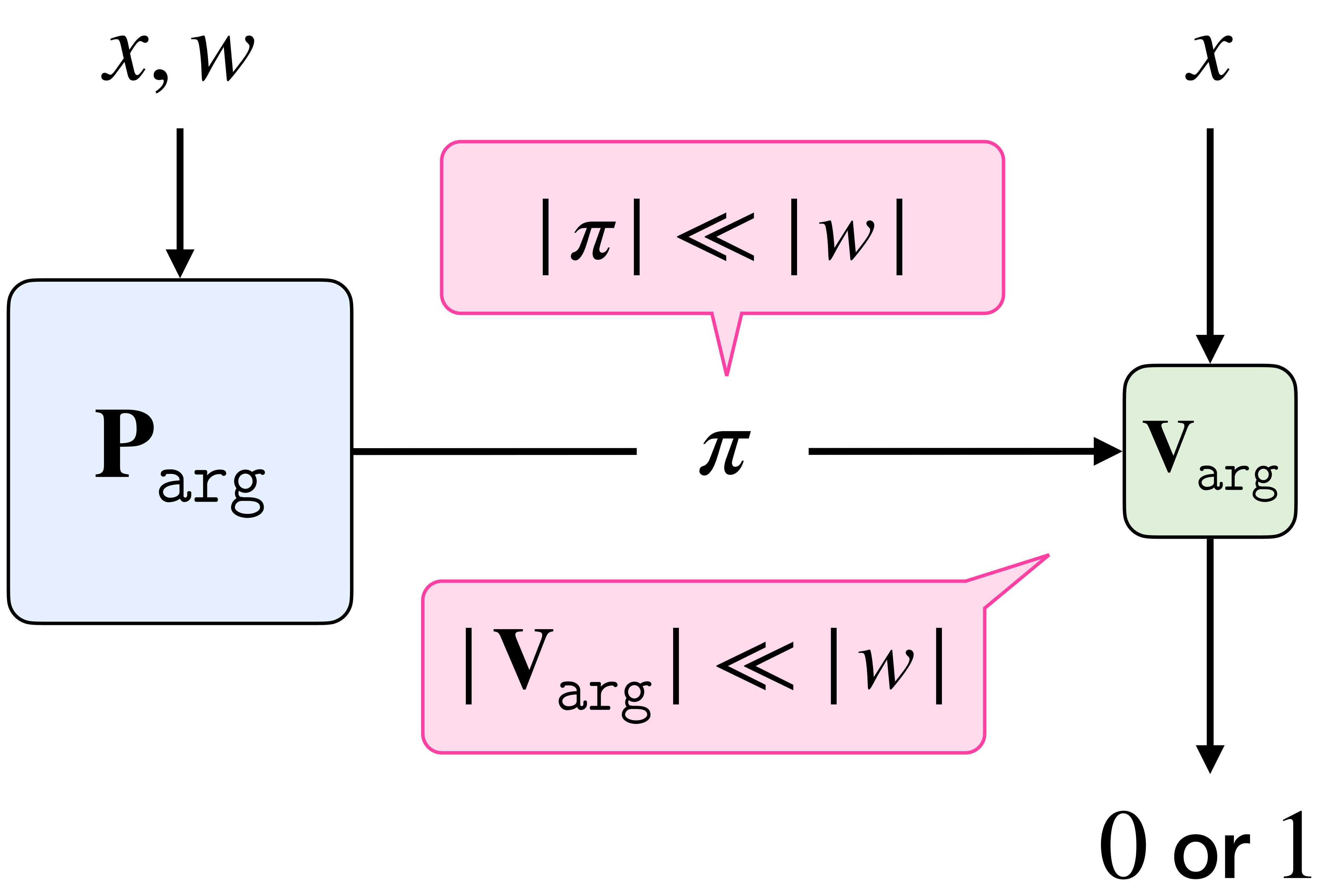
*No for IVC for P [DGKV22, PP23]

Accumulation Schemes

Review: SNARGs

succinct non-interactive arguments

$$(x, w) \in_? R$$



completeness

if $(x, w) \in R$

then $\mathbf{V}_{\texttt{arg}} \to 1$

$L(R) := \{x : \exists w, (x, w) \in R\}$

soundness

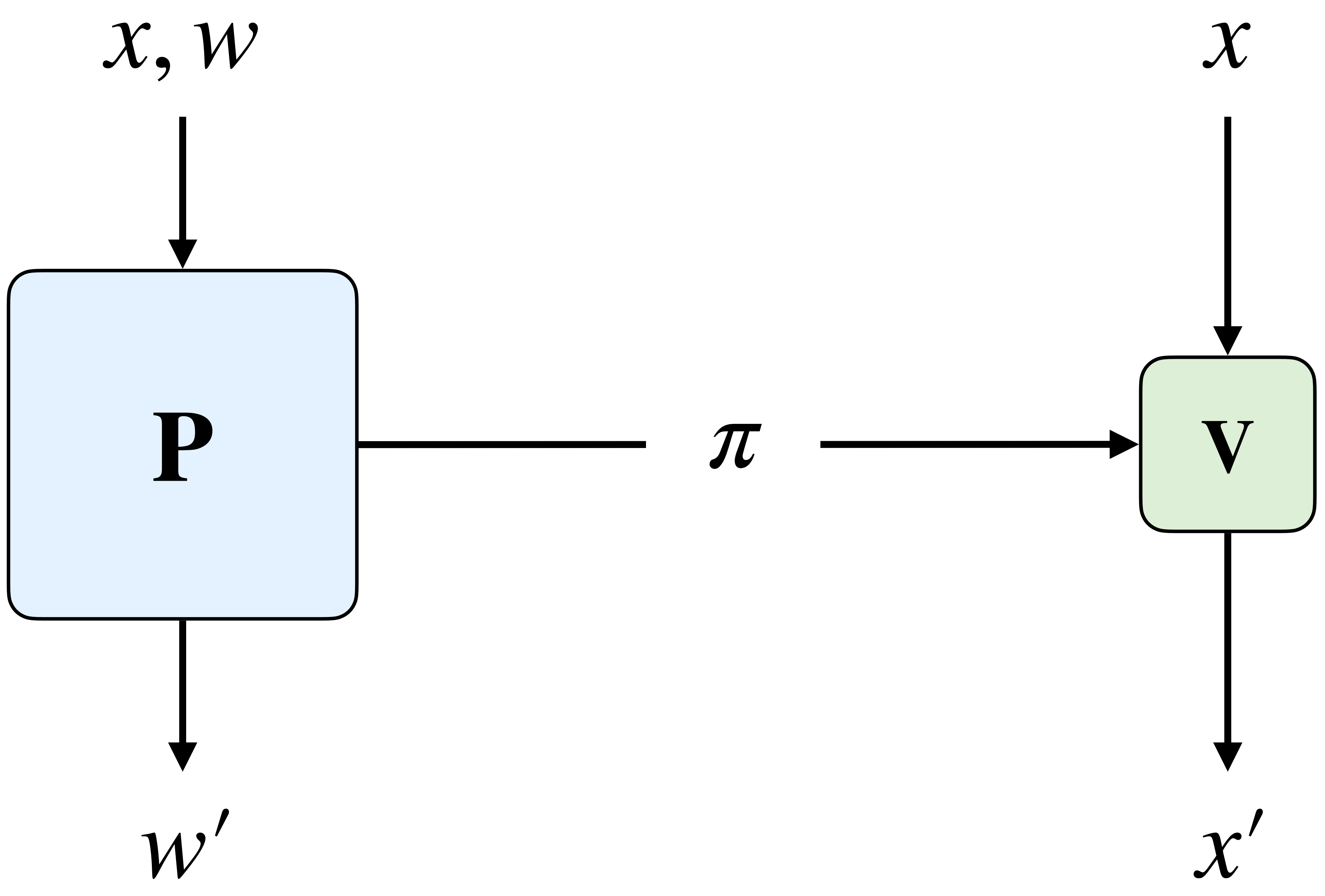
if $x \notin L(R)$

then w.h.p. $\mathbf{V}_{\texttt{arg}} \to 0$

in general: knowledge soundness

Background: reductions [KP22]

$$(x, w) \in_? R$$



$$(x', w') \in_? R'$$

completeness

if $(x, w) \in R$

then $(x', w') \in R'$

soundness

if $x \notin L(R)$

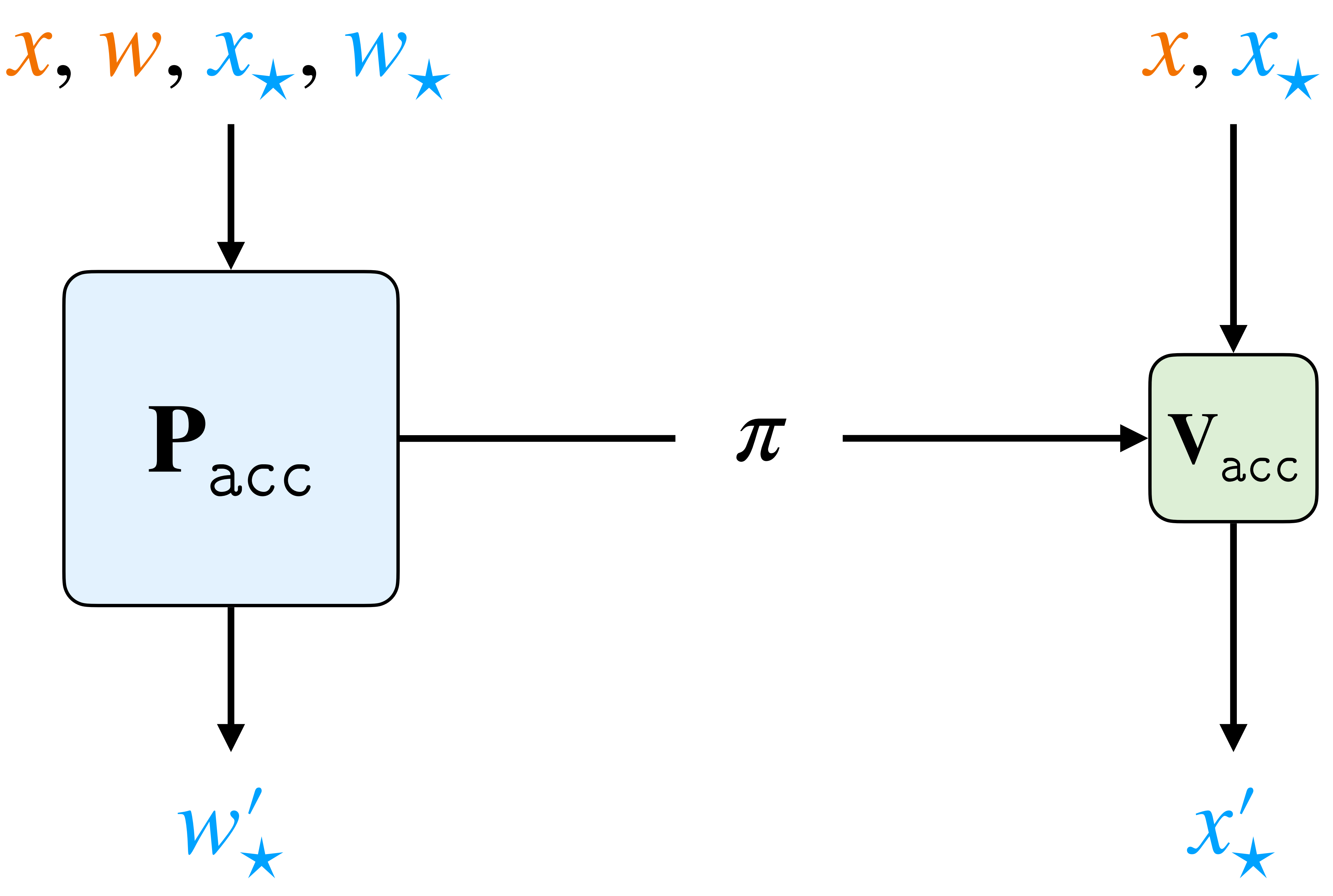
then w.h.p. $x' \notin L(R')$

in general: knowledge soundness

Accumulation schemes

[BGH19,BCMS20,BCLMS21,KST22]

$$(x, w) \in_? R \;\wedge\; (x_\star, w_\star) \in_? R_\star$$

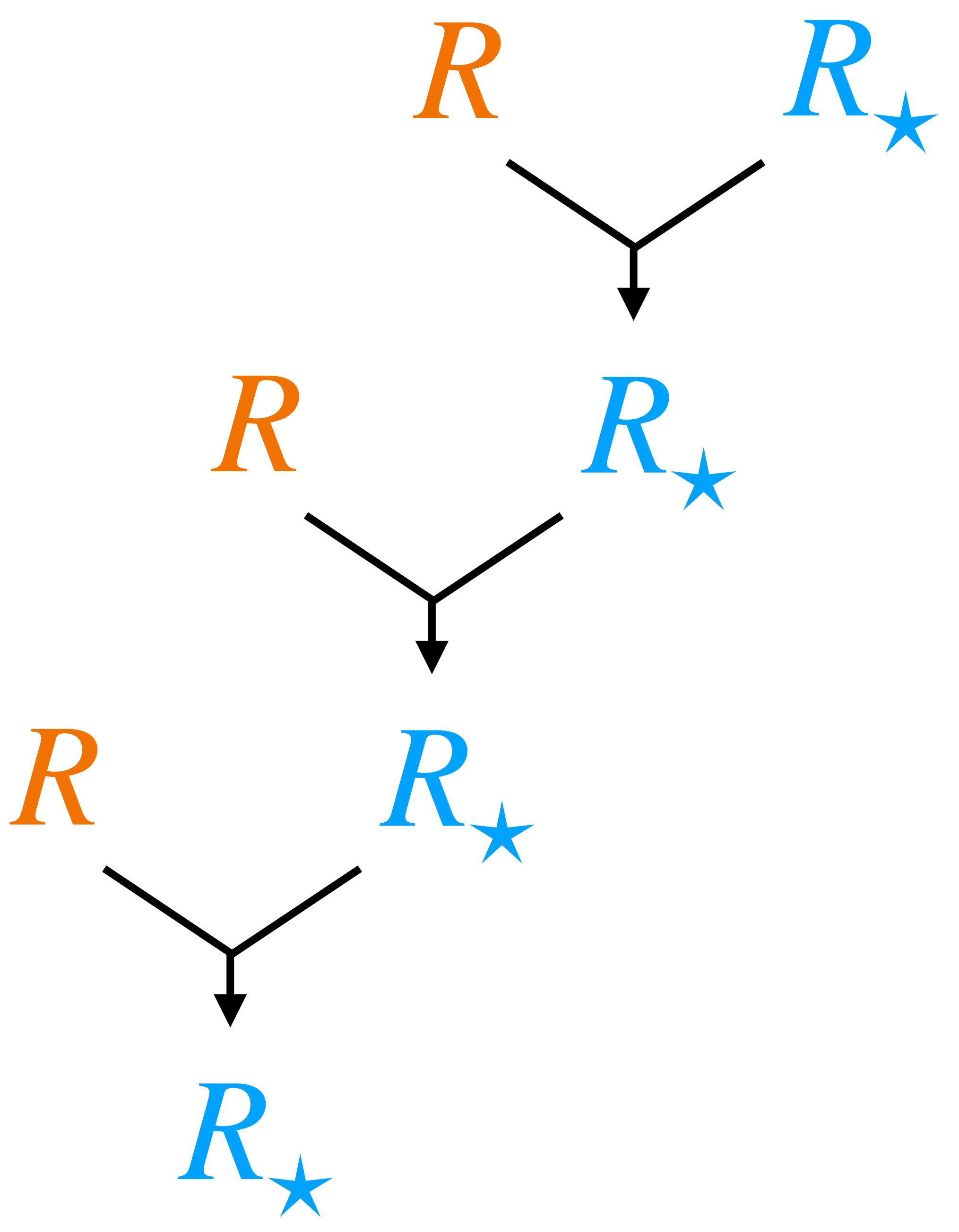


$$(x'_\star, w'_\star) \in_? R_\star$$

in general: $R^n \times R_\star^m \to R_\star$

accumulation scheme for R:

reduction from $R \times R_\star$ to $R_\star$



more precisely, a split accumulation (or folding) scheme [BCLMS21], [KST22]

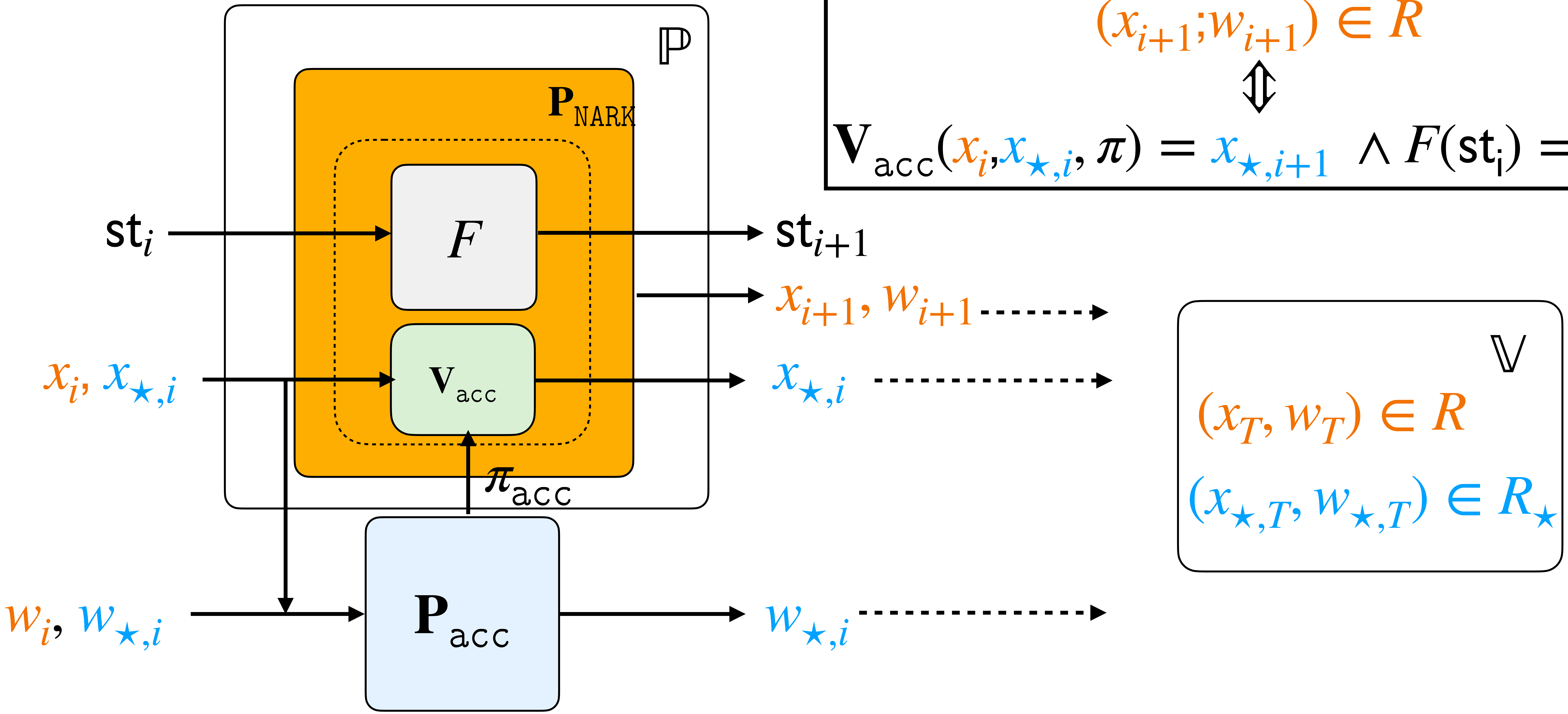
weaker than an argument for R!

IVC from accumulation [BCLMS20]

$\mathbf{P}_{\mathtt{NARK}}$ outputs $(x, w) \in R$

$\mathbf{V}_{\mathtt{NARK}}$ checks $(x, w) \in R$

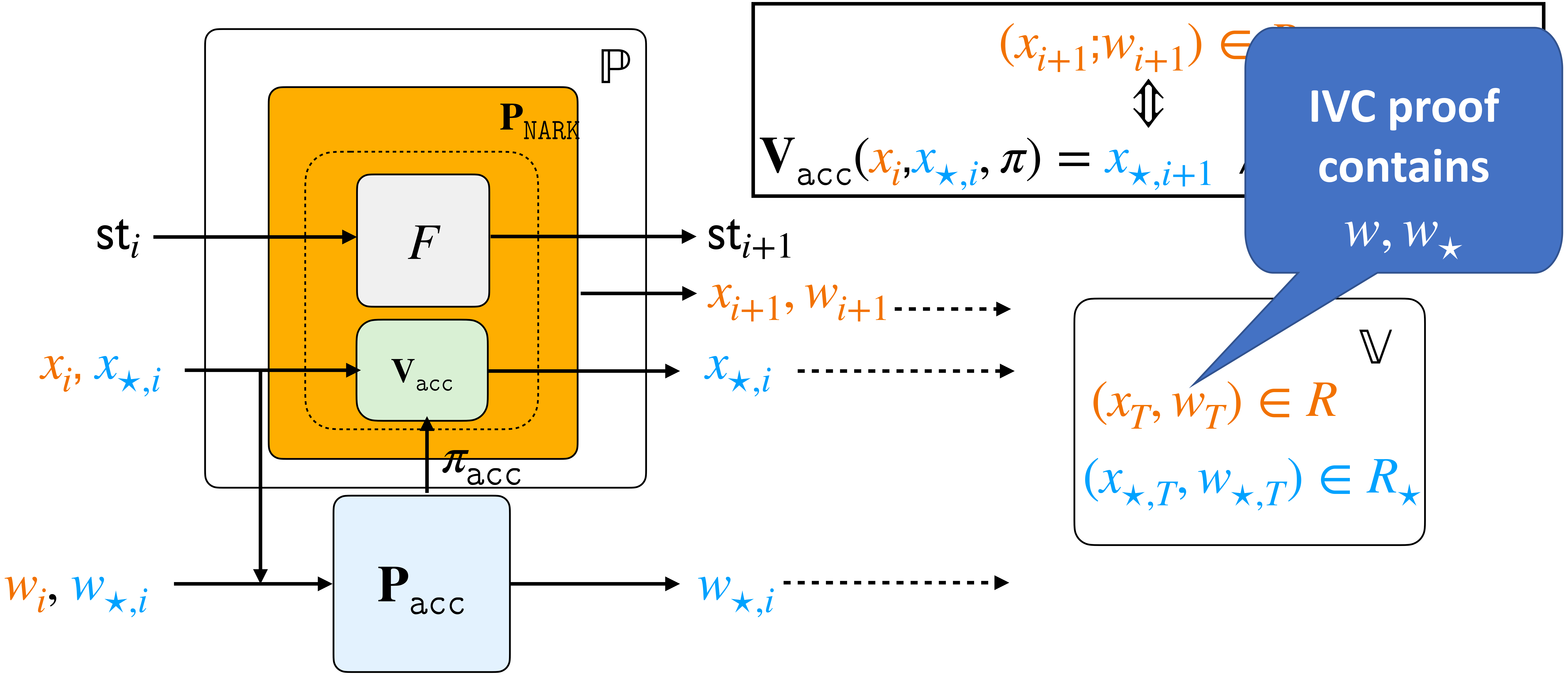
$$(x_{i+1}; w_{i+1}) \in R$$
$$\Updownarrow$$
$$\mathbf{V}_{\mathsf{acc}}(x_i, x_{\star,i}, \pi) = x_{\star,i+1} \;\wedge\; F(\mathsf{st}_i) = \mathsf{st}_{i+1}$$



IVC from accumulation [BCLMS20]

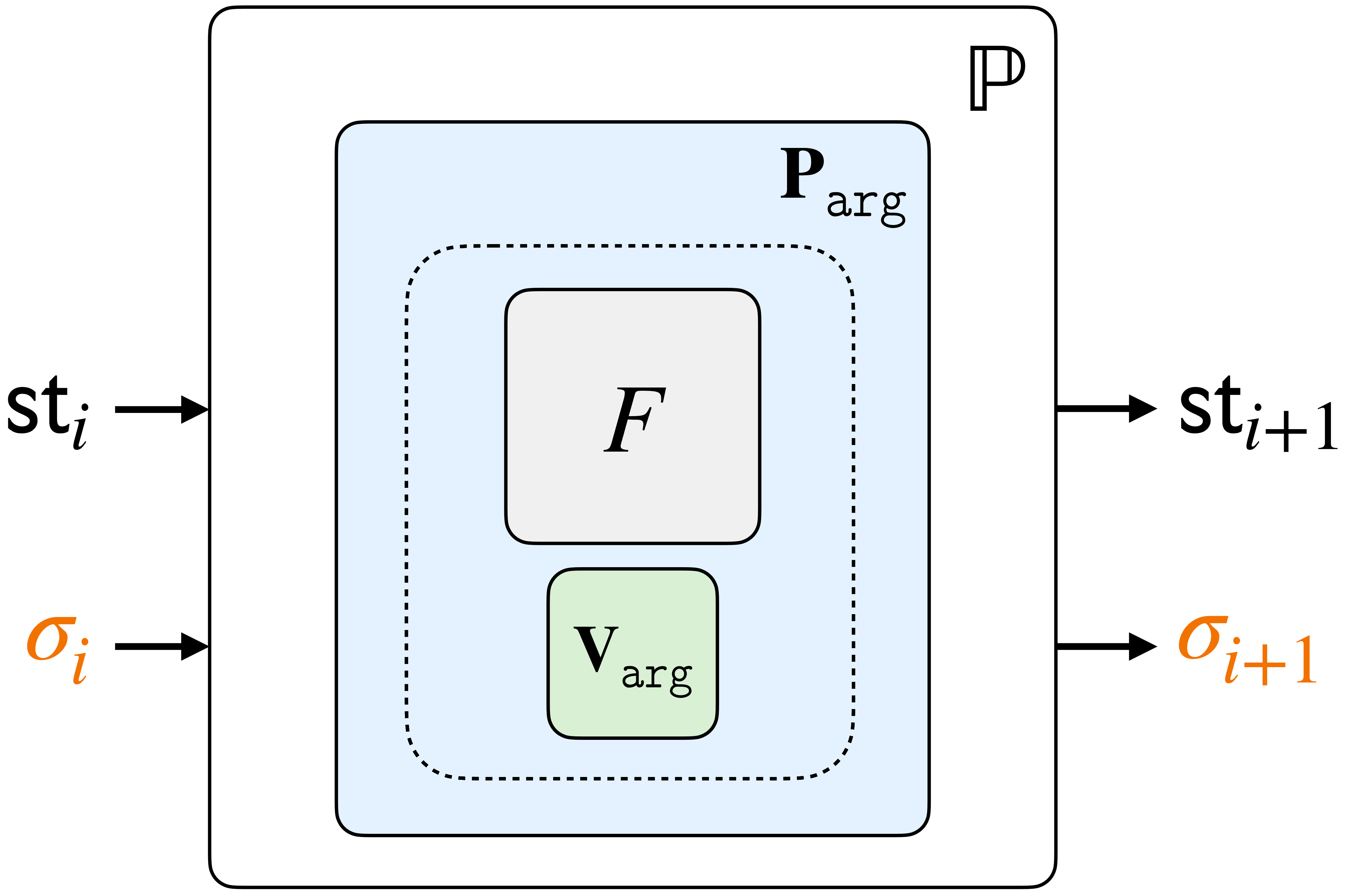
$\mathbf{P}_{\texttt{NARK}}$ outputs $(x, w) \in R$

$\mathbf{V}_{\texttt{NARK}}$ checks $(x, w) \in R$

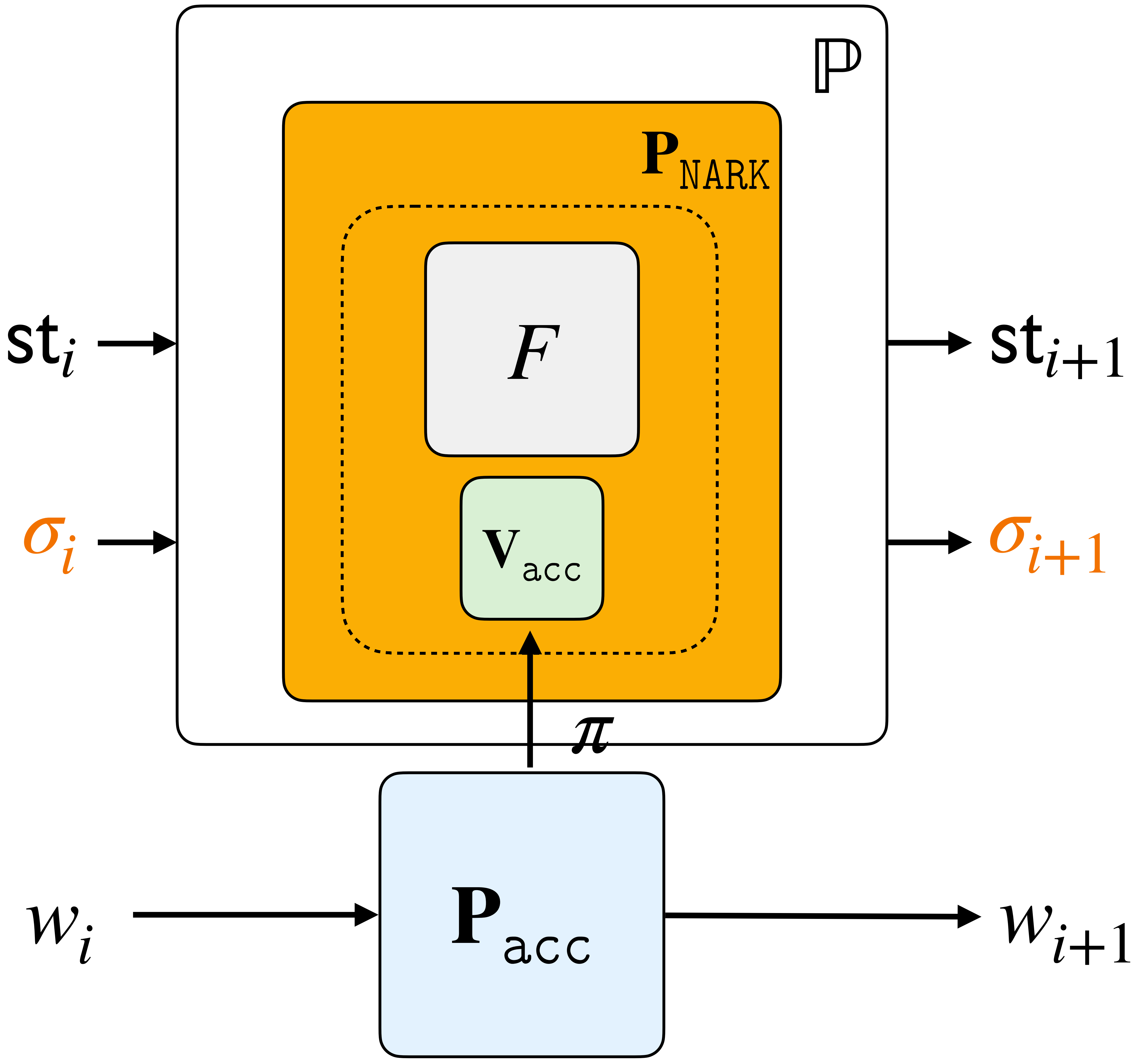


Why accumulate?

IVC from succinct arguments



IVC from accumulation



$\mathbf{P}_{\texttt{acc}} + \mathbf{P}_{\texttt{NARK}}$ can be faster than $\mathbf{P}_{\texttt{arg}}$

$\mathbf{V}_{\texttt{acc}}$ can be smaller than $\mathbf{V}_{\texttt{arg}}$

Accumulation and SNARKs

- Accumulation is simpler than SNARKs
  - We can construct it in settings and with efficiencies that don’t admit SNARKs
- Accumulation suffices to build IVC/PCD
- IVC/PCD enables building SNARKs
  - Set F to be a step function of a VM
- How can this be?
  - All known “interesting” accumulation schemes require random oracles
  - To build IVC/PCD we need accumulation in standard models (heuristic jump)

Accumulation is "easy" [BCLMS21, KST22]

Check: $\mathbf{a}, \mathbf{b}, \mathbf{c}$ s.t. $\mathbf{a}_i + \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\mathrm{NARK}}$ $\longrightarrow x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\mathsf{P}_{\mathrm{acc}}$ $\quad$ V_{acc}

$$\xleftarrow{\alpha \leftarrow \mathbb{F}}$$

$$(x, w) \in R : \{x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \wedge \mathbf{a} + \mathbf{b} - \mathbf{c} = \mathbf{0}\}$$

Commit can be built from DLOG

$\text{Commit}(w) + \text{Commit}(w') = \text{Commit}(w + w')$

$$(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')$$

$$(x, w) \in R, (x, w') \in R \Rightarrow (x, w) + Y \cdot (x', w') \in R$$

Accumulation is "easy" [BCLMS21, KST22]

Check: **a**, **b**, **c** s.t. $\mathbf{a}_i + \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\mathtt{NARK}}$ $\longrightarrow x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\mathrm{P}_{\mathrm{acc}}$ $\quad V_{\mathrm{acc}}$

$\alpha \leftarrow \mathbb{F}$

Non-interactivity through Fiat-Shamir

$, w) \in R :$

$, \mathbf{b}, \mathbf{c}) \wedge \mathbf{a} + \mathbf{b} - \mathbf{c} = \mathbf{0}\}$

$$(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')$$

Commit can be built from DLOG

$\text{Commit}(w) + \text{Commit}(w') = \text{Commit}(w + w')$

$$(x, w) \in R, (x, w') \in R \Rightarrow (x, w) + Y \cdot (x', w') \in R$$

Accumulation is "easy" [BCLMS21, KST22]

Check: $\mathbf{a}, \mathbf{b}, \mathbf{c}$ s.t. $\mathbf{a}_i \cdot \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\texttt{NARK}}$ $\longrightarrow x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

P_{acc} $\qquad$ V_{acc}

$\text{P}_{\text{acc}} \xrightarrow{\text{ct}} V_{\text{acc}}$

$V_{\text{acc}} \xrightarrow{\alpha} \text{P}_{\text{acc}}$

$$(x, w) \in R : \{x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \wedge \mathbf{a} \circ \mathbf{b} - \mathbf{c} = \mathbf{0}\}$$

$$x'' \leftarrow x + \alpha \cdot x' || \text{ct}$$
$$w'' \leftarrow w + \alpha \cdot w'$$
$$x'', w'' \in R_\star$$

$$(x, w) \in R_\star : \{x = \text{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \text{ct}) \wedge \mathbf{a} \circ \mathbf{b} - \mathbf{c} = \text{ct}\}$$

$$(x, w) \in R, (x, w') \in R \nRightarrow (x'', w'') \in R$$

but

$$(\mathbf{a} + Y \cdot \mathbf{a}') \circ (\mathbf{b} + Y \cdot \mathbf{b}') = \mathbf{c} + Y^2 \cdot \mathbf{c}' + Y \cdot \text{ct}$$

Accumulation for multiplication

- Reduction from $R \times R \to R_\star$
- Reduction from $R \times R_\star \to R_\star$ is very similar!
- Multiplication and addition suffice to build accumulation for NP
- Only cryptography needed is a homomorphic vector commitment + Fiat-Shamir
  - No PCPs
  - No polynomial commitments
  - **No trusted setup**
  - **Single commitment**
- Acc verifier does **2 group scalar multiplications** (check homomorphism)
  - Needs to check elliptic curve operations
  - In practice: Use cycles of elliptic curves for efficiency (mismatched fields)

A universe of accumulation

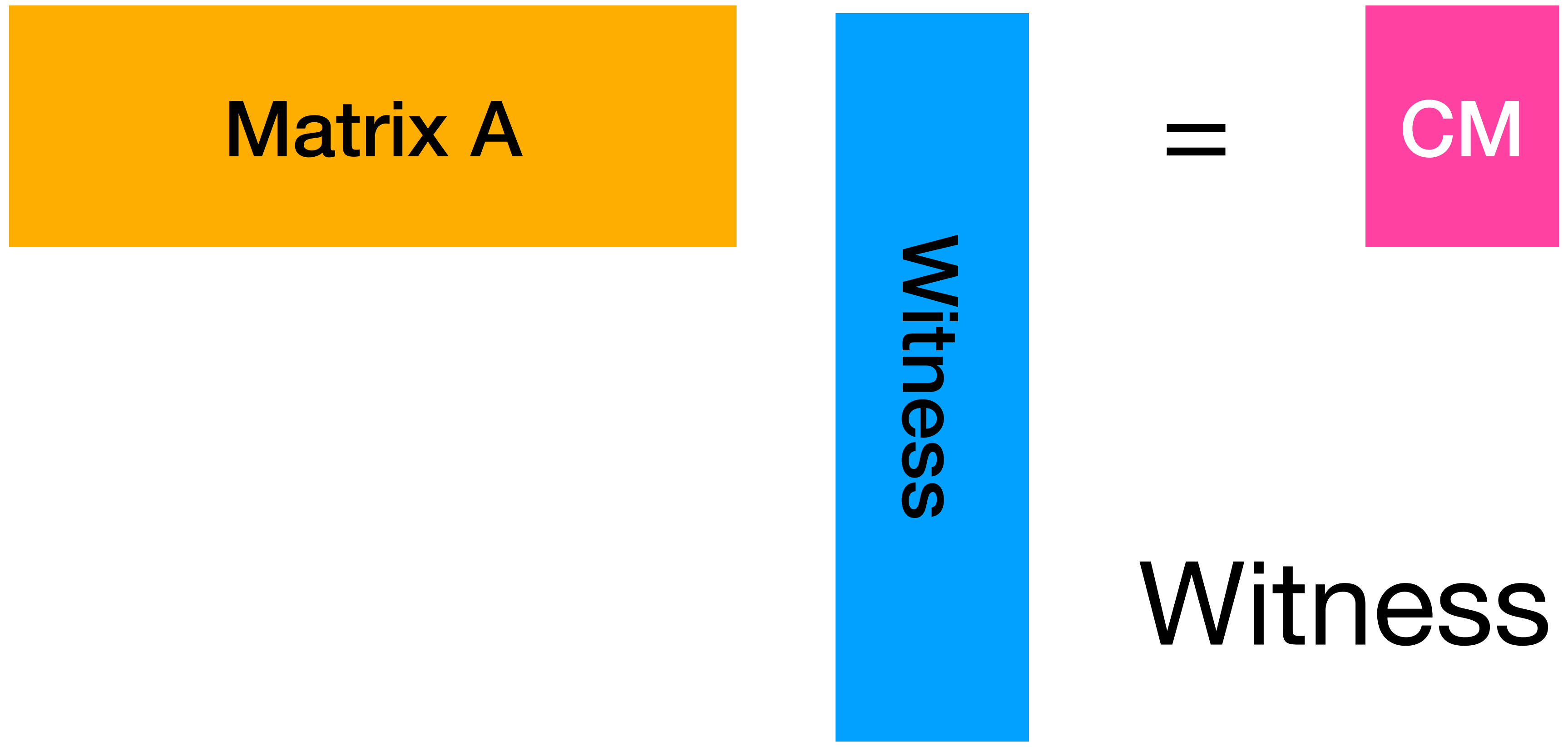
- Lowering recursion overhead [KotSetSzi22,KotSet23,BünChe23,DimGarManVla24,Bün24]
  - Down to only one scalar multiplication
  - less than 10k gates vs. 100k+ gates for SNARKs
- Multi-instance proving (for PCD)[KotSet23,EagGab23]
- Supporting high degree gates [Moh22,KotSet23,BC23]
- Faster prover[KotSet24]
- Handling cycles of elliptic curves[KotSet23b]
- Zero-Knowlege support [ZheGaoGuoXia23]
- Memory operations [BC24,AruSet24]
- Outsourcing verification [ZSCZ25]
- Smaller accumulators [BGH19,BF24,KZHB25]
- Non-uniformity[KS22,BC23,KZHB25]
- Parallel SNARK constructions [NDTCB24]
- Tighter security analysis [NBS23,LS24]
- …

Post-quantum accumulation

- Accumulation verifier needs to check $\overline{w}' \stackrel{?}{=} \overline{w} + \alpha \cdot \overline{z}$
- Accumulation scheme require homomorphic vector commitment
  - Pedersen commitment is built from the DLOG assumption
  - Not post-quantum
- **Goal:** Get rid of the homomorphism

Lattice-based accumulation

- SIS-commitment



Witness needs to be low-norm

- Only limited homomorphism
- Idea: Resplit witness and combine low-norm components [BC24]
- Multiple improvements [GKNP24,BC25,SN25] (See Binyi's talk)
- Larger recursion overhead than EC-based, but possibly very fast prover

Can we build accumulation in the RO?

- No additional assumptions
- Trivial answer: Yes, SNARKs imply accumlation
- Can we do better?

Homomorphic accumulation

Check: **a**, **b**, **c** s.t. $\mathbf{a}_i + \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\mathrm{NARK}}$ $\longrightarrow x = \mathrm{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\mathrm{P}_{\mathrm{acc}}$ $\qquad\qquad$ V_{acc}

$$\xleftarrow{\alpha \leftarrow \mathbb{F}}$$

$$\boxed{(x, w) \in R : \{x = \mathrm{Commit}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \wedge \mathbf{a} + \mathbf{b} - \mathbf{c} = \mathbf{0}\}}$$

$$\boxed{(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')}$$

Commit can be built from DLOG

$\mathrm{Commit}(w) + \mathrm{Commit}(w') = \mathrm{Commit}(w + w')$

$$\boxed{(x, w) \in R, (x, w') \in R \Rightarrow (x, w) + Y \cdot (x', w') \in R}$$

Non-Homomorphic accumulation

Check: **a**, **b**, **c** s.t. $\mathbf{a}_i + \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\text{NARK}}$ $\longrightarrow x = \mathsf{MT}(\mathbf{a}, \mathbf{b}, \mathbf{c})$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

P_{acc} $\qquad$ V_{acc}

$$\xleftarrow{\alpha \leftarrow \mathbb{F}}$$

$$(x, w) \in R : \{x = \mathsf{MT}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \wedge \mathbf{a} + \mathbf{b} - \mathbf{c} = \mathbf{0}\}$$

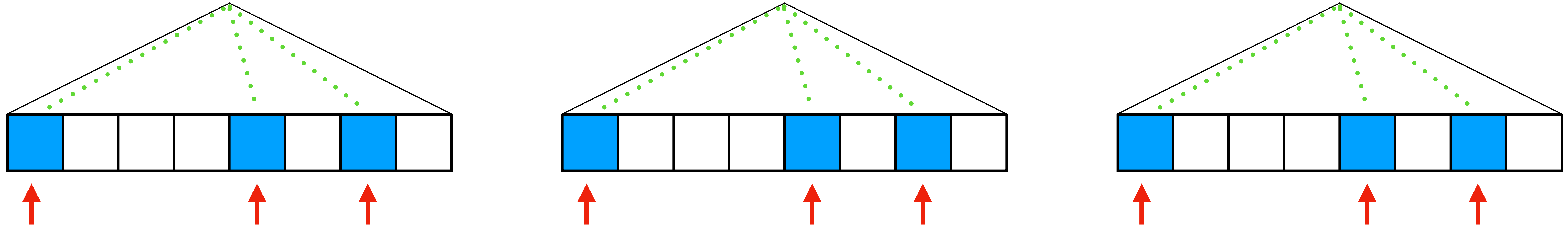
$$(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')$$

V_{acc} **can't check this operation anymore**

$$(x, w) \in R, (x, w') \in R \Rightarrow (x, w) + Y \cdot (x', w') \in R$$

Checking the homomorphism

$$\overline{w}' \stackrel{?}{=} \overline{w} + \alpha \cdot \overline{z}$$



P

Sample $Q \subset [n]$ V

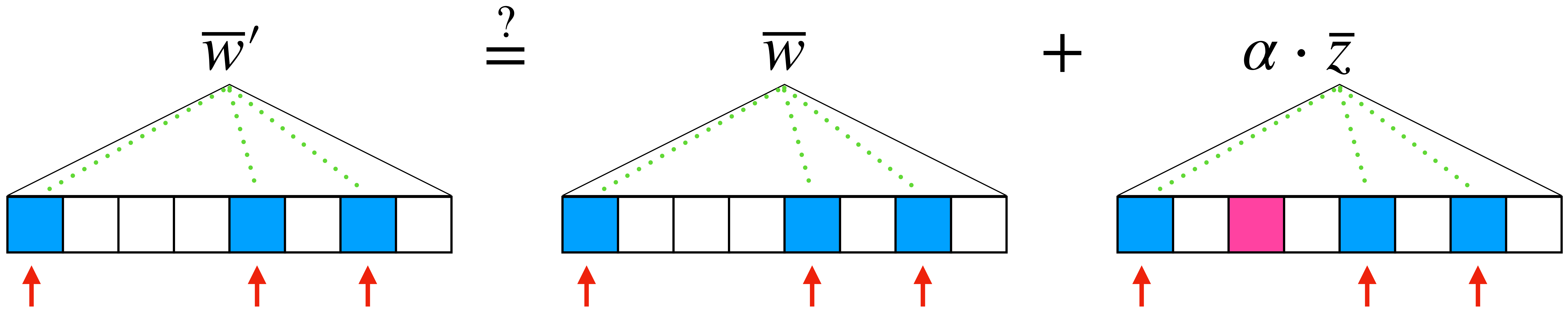
$\xleftarrow{\quad Q \quad}$

$w'[Q], w[Q], z[Q]$
opening proofs
$\xrightarrow{\qquad\qquad}$

Check opening proofs

For each $i \in Q$:
$w'[i] \stackrel{?}{=} w[i] + \alpha \cdot z[i]$

Checking the homomorphism

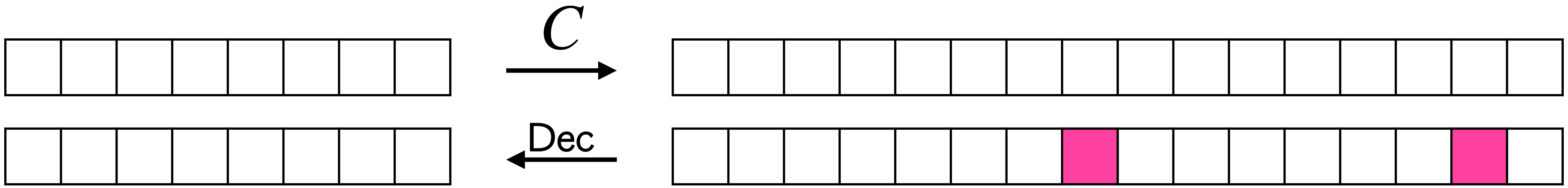


- Suppose δ-fraction of locations are inconsistent
- Then t queries miss w.p. $(1-\delta)^t$
- $t = \dfrac{\lambda}{\delta} \implies (1-\delta)^t \leq 2^{-\lambda}$

Problem: How to detect a single inconsistency?

New tool: linear codes

- Linear map $C : \mathbb{F}^n \to \mathbb{F}^\ell$ from "messages" to "codewords"
- Distance: minimum relative Hamming distance between any two codewords
- Decoding: given a noisy codeword, recover the original message



- Unique decoding radius = maximum number of errors allowed
- We want a linear code with large distance/decoding radius, e.g. Reed–Solomon codes

Attempt 1: Spotchecks [BMNW24]

Check: $\mathbf{a}, \mathbf{b}, \mathbf{c}$ s.t. $\mathbf{a}_i + \mathbf{b}_i = \mathbf{c}_i \in \mathbb{F}$ for all $i \in [n]$

$\mathbf{P}_{\text{NARK}}$ $\longrightarrow x = \mathsf{MT}(C(\mathbf{a}, \mathbf{b}, \mathbf{c}))$

$\longrightarrow w = (\mathbf{a}, \mathbf{b}, \mathbf{c})$

P_{acc} $\qquad$ V_{acc}

$$\xleftarrow{\alpha \leftarrow \mathbb{F}}$$

$$(x, w) \in R: \{x = \mathsf{MT}(C(\mathbf{a}, \mathbf{b}, \mathbf{c})) \wedge \mathbf{a} + \mathbf{b} - \mathbf{c} = \mathbf{0}\}$$

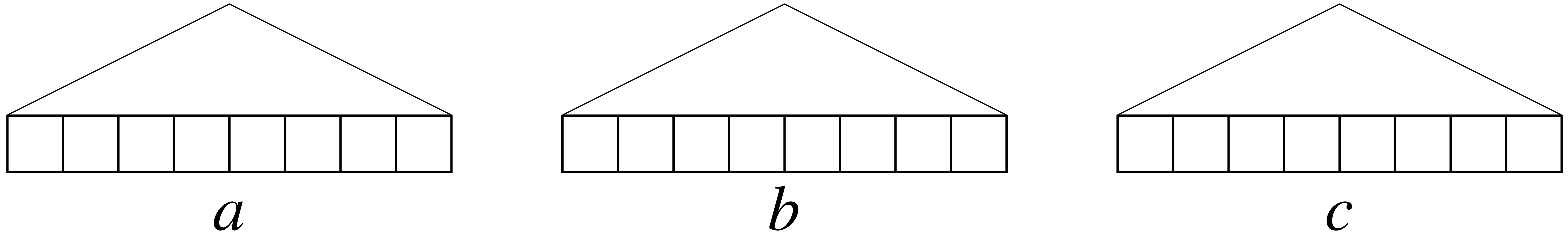
$$\boxed{(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')}$$

Use linear code C
Check homomorphism at
Random spots

$$\boxed{(x, w) \in R, (x, w') \in R \Rightarrow (x, w) + Y \cdot (x', w') \in R}$$

Soundness analysis

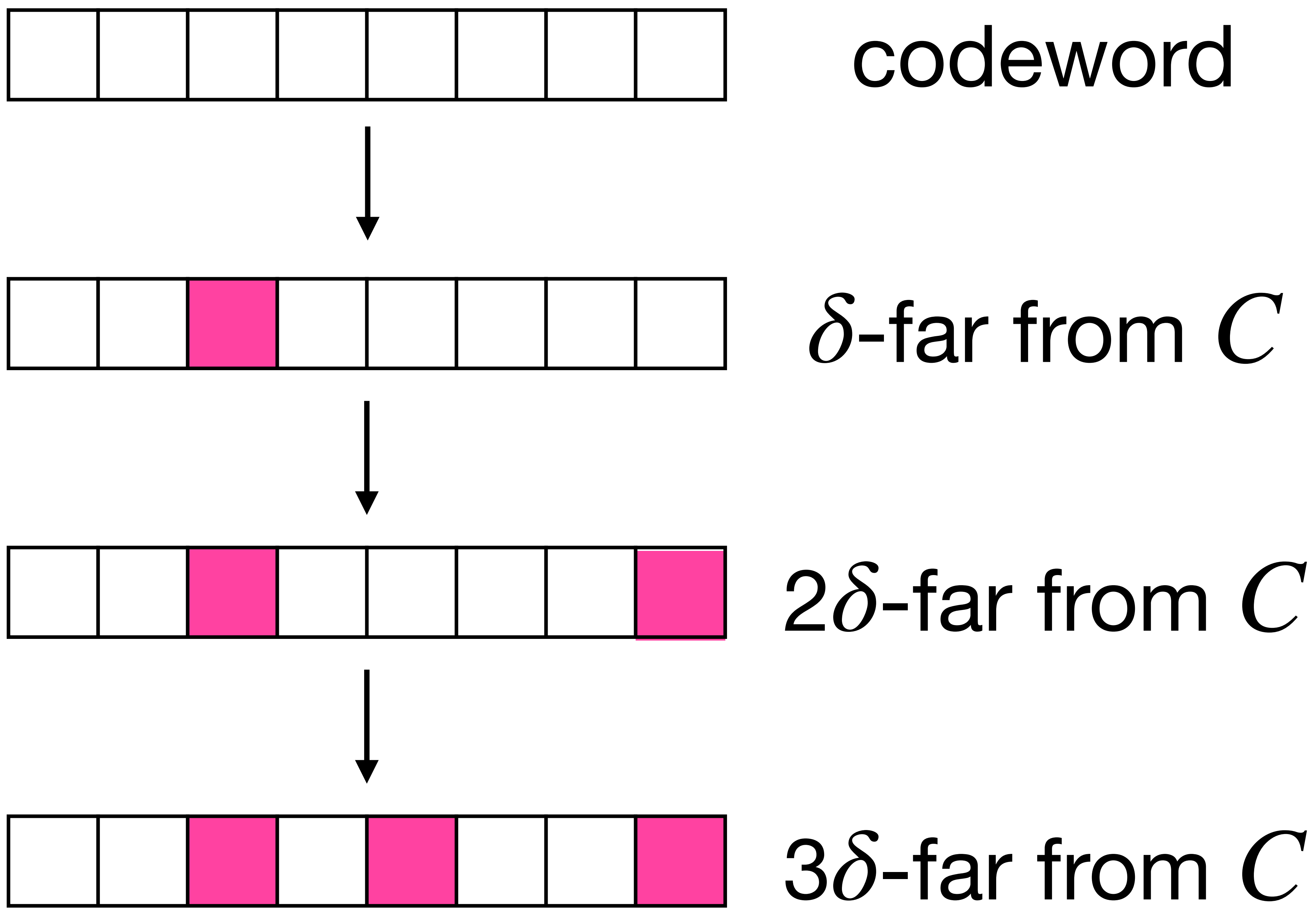
$$\overline{w}' \stackrel{?}{=} \overline{w} + \alpha \cdot \overline{z}$$



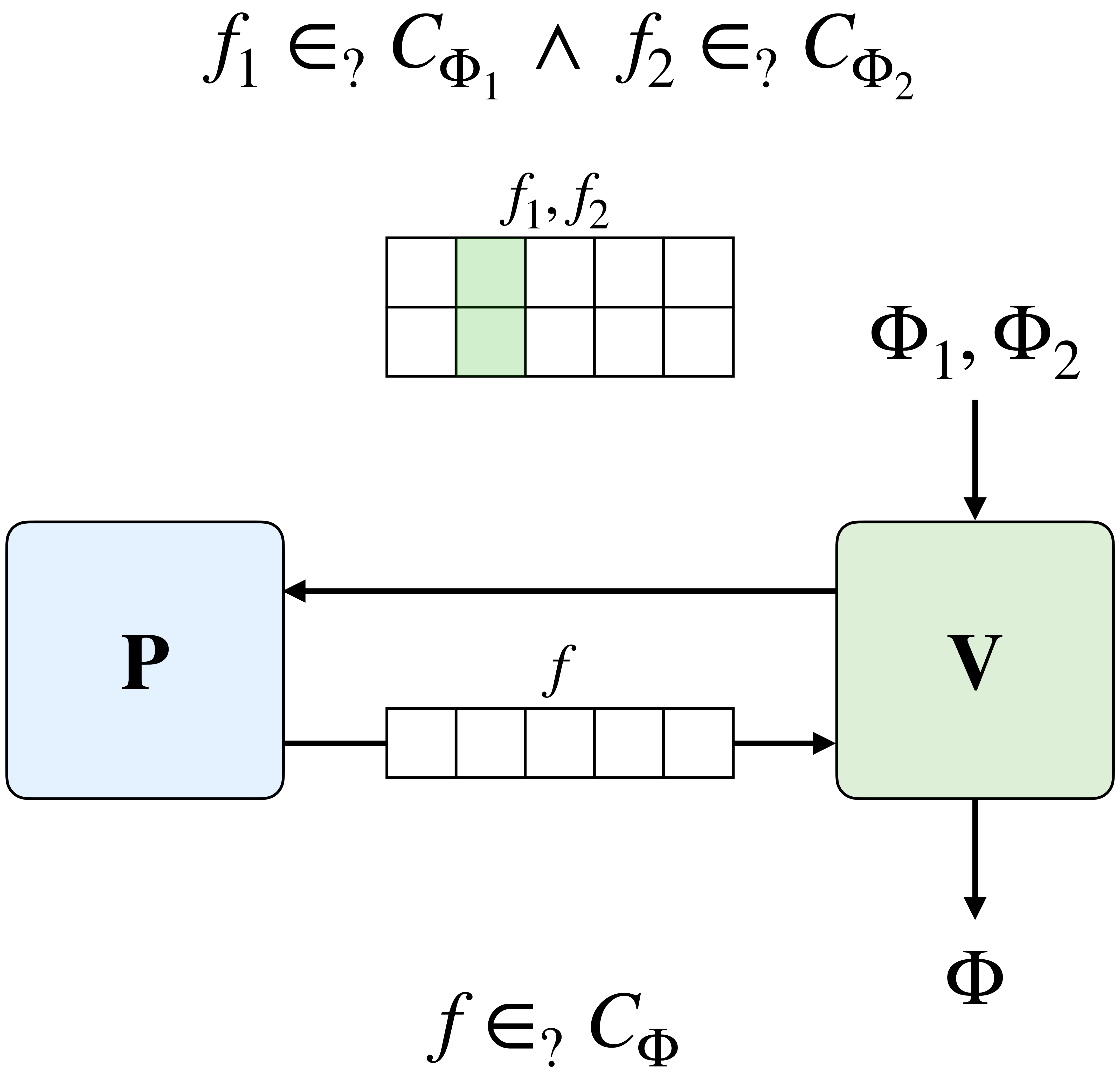
- Decider guarantees: $a \in C$
- Verifier guarantees: $\Delta(a, b + \alpha c) < \delta \implies b + \alpha c$ is δ-close to C
- b are c are δ-close to C (by proximity gap for C: BCIKS23, RVW13, AHIV17, DP23a)
- $\implies$ Dec(a) = Dec$(b) + \alpha \cdot$ Dec(c)
    - "Proof": Encode both sides, they are 3δ-close → equal (assuming $3\delta <$ distance)

Accumulating multiple times

- To support d accumulations:
    - Spot check parameter δ
    - $d\delta <$ unique decoding radius
- **Matching attack**



Solution: Use constrained codes [BMNW24,KNS24,Szep24,BCFW25]



Any linear code $C : \mathbb{F}^k \to \mathbb{F}^n$

Any "low-degree polynomial" constraint $\Phi : \mathbb{F}^k \to \mathbb{F}$

Constrained code
$C_\Phi := \{C(v) : \Phi(v) = 0\}$

Accumulation from constrained codes

- Prover sends new claimed codeword f
- Verifier queries f_1, f_2 at random locations
- Constrain f given the query responses from f_1, f_2

$$\boxed{\text{Lemma: If } \Delta(f_1, C_\phi) > \delta \vee \Delta(f_2, C_\phi) > \delta \underset{\text{w.h.p}}{\implies} \Delta(f, C_\phi) > \delta}$$

- More details: William's talk

Accumulation for linear codes[BCPFW25,BMMS25]

- Accumulation for **any** linear code with essentially **optimal** parameters
  - Accumulation verifier does O(λ) oracle queries (MT paths after compilation)
  - Not known for SNARKs
  - Linear time prover (for large fields)
  - Up to list-decoding radius
- Straightline extraction without efficient decoding
- Direct accumulation for NP
  - No need to go through PIOPs

Constructions open questions

- Linear time accumulation for small fields (binary even)
  - Easier than the related SNARK question
- Linear time accumulation from lattices
  - Smaller acc verifier than hash-based schemes
- Accumulation without random oracles
  - Would yield PCD and SNARK in the standard model
  - Minimal assumptions needed?
- Smaller accumulator size
  - $\texttt{acc}.w$ needs to be forwarded as part of the PCD
  - For all post-quantum constructions $|\texttt{acc}| = \Theta(|F|)$. High communication
  - Can we do better?

Recursive proofs are powerful but can be built from simple assumptions*

* with security jumps

Thank you

Citations (general)

- [Val08] Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency
- [CT10] Proof-Carrying Data and Hearsay Arguments from Signature Cards
- [BCCT13] Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data
- [COS20] Fractal: Post-Quantum and Transparent Recursive Proofs from Holography
- [KP22] Algebraic Reductions of Knowledge
- [DGKV22] Rate-1 non-interactive arguments for batch-NP and applications
- [PP23] Incrementally Verifiable Computation via Rate-1 Batch Arguments

Citations (applications)

- [CTV13] Enforcing Language Semantics Using Proof-Carrying Data
- [CTV15] Cluster Computing in Zero Knowledge
- [NT16] PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations
- [BCCGMW18] Zexe: Enabling Decentralized Private Computation
- [BCG20] Breaking the $O(\sqrt{n})$-Bit Barrier: Byzantine Agreement with Polylog Bits Per Party
- [KB20] Proof of Necessary Work: Succinct State Verification with Fairness Guarantee
- [BMRS20] Coda: Decentralized Cryptocurrency at Scale
- [BBBF20] Verifiable Delay Functions
- [CCDW20] Reducing Participation Costs via Incremental Verification for Ledger Systems
- [KB20] Proof of Necessary Work: Succinct State Verification with Fairness Guarantees

Citations (security)

- [CGH98] The Random Oracle Methodology Revisited
- [Bar01] How to go beyond the black-box simulation barrier
- [GK03] On the (In)security of the Fiat-Shamir Paradigm
- [CCS22] On succinct non-interactive arguments in relativized worlds
- [Zhandry22] Augmented Random Oracles
- [CCGOS23] Proof-Carrying Data From Arithmetized Random Oracles
- [CGSY23] Security Bounds for Proof-Carrying Data from Straightline Extractors
- [BCG24] Relativized Succinct Arguments in the ROM Do Not Exist
- [RT24] Straight-Line Knowledge Extraction for Multi-Round Protocols
- [KRS25] How to Prove False Statements: Practical Attacks on Fiat-Shamir
- [AY25] Towards a White-Box Secure Fiat-Shamir Transformation

Citations (accumulation)

- [BGH19] Halo: Recursive Proof Composition without a Trusted Setup
- [BCMS20] Recursive Proof Composition from Accumulation Schemes
- [BCLMS21] Proof-Carrying Data Without Succinct Arguments
- [KST22] Nova: Recursive Zero-Knowledge Arguments from Folding Schemes
- [Moh22] Sangria
- [KS22] SuperNova
- [KotSet23] HyperNova
- [BC23] Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols
- [EatGab23] Protogalaxy
- [KotSet23b] Cyclefold
- [ZheGaoGuoXia23] KiloNova
- [NBS23] Revisiting the Nova Proof System on a Cycle of Curves
- [BF24] Mira
- [DimGarManVla24] Mova
- [Bünz24] Ova
- [KotSet24] NeutronNova
- [BC24] Proofs for deep thought
- [BC24b] LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems
- [AruSet24] Nebula
- [NDCTB24] Mangrove: A Scalable Framework for Folding-Based SNARKs
- [LS24] On the Security of Nova Recursive Proof System
- [GKNP24] Lova
- [BMNW24] Accumulation without homomorphism
- [BC25] Latticefold+: Faster, Simpler, Shorter Lattice-Based Folding for Succinct Proof Systems
- [BCFW25] Linear-Time Accumulation Schemes
- [KZHB25] KZH-Fold: Accountable Voting from Sublinear Accumulation
- [ZSCZ25] MicroNova
- [SN25] Neo