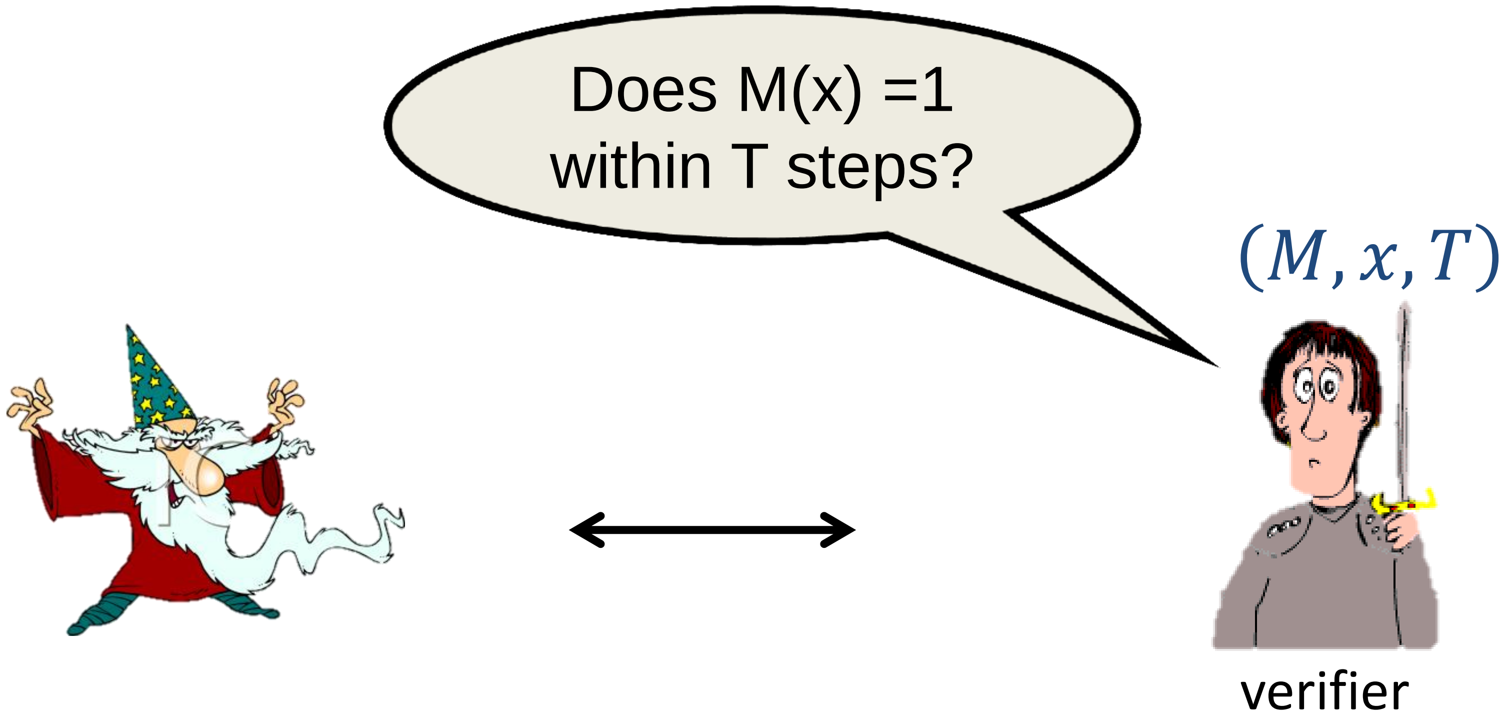
How to Bootstrap a SNARK *in Public*

Nir Bitansky, Ran Canetti, Alessandro Chiesa, Eran Tromer

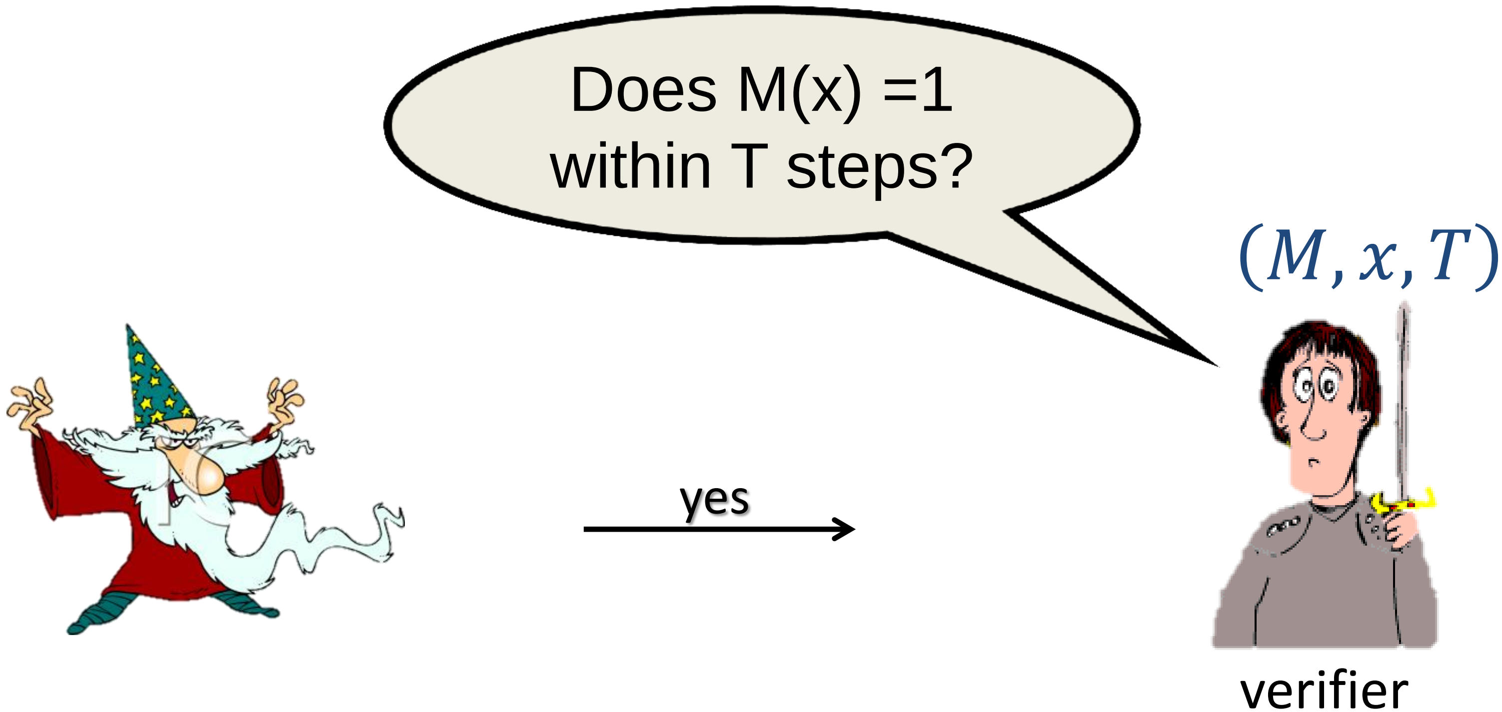
How quickly can we verify the result of long computations?

How quickly can we verify the result of long computations?

(Plain version)



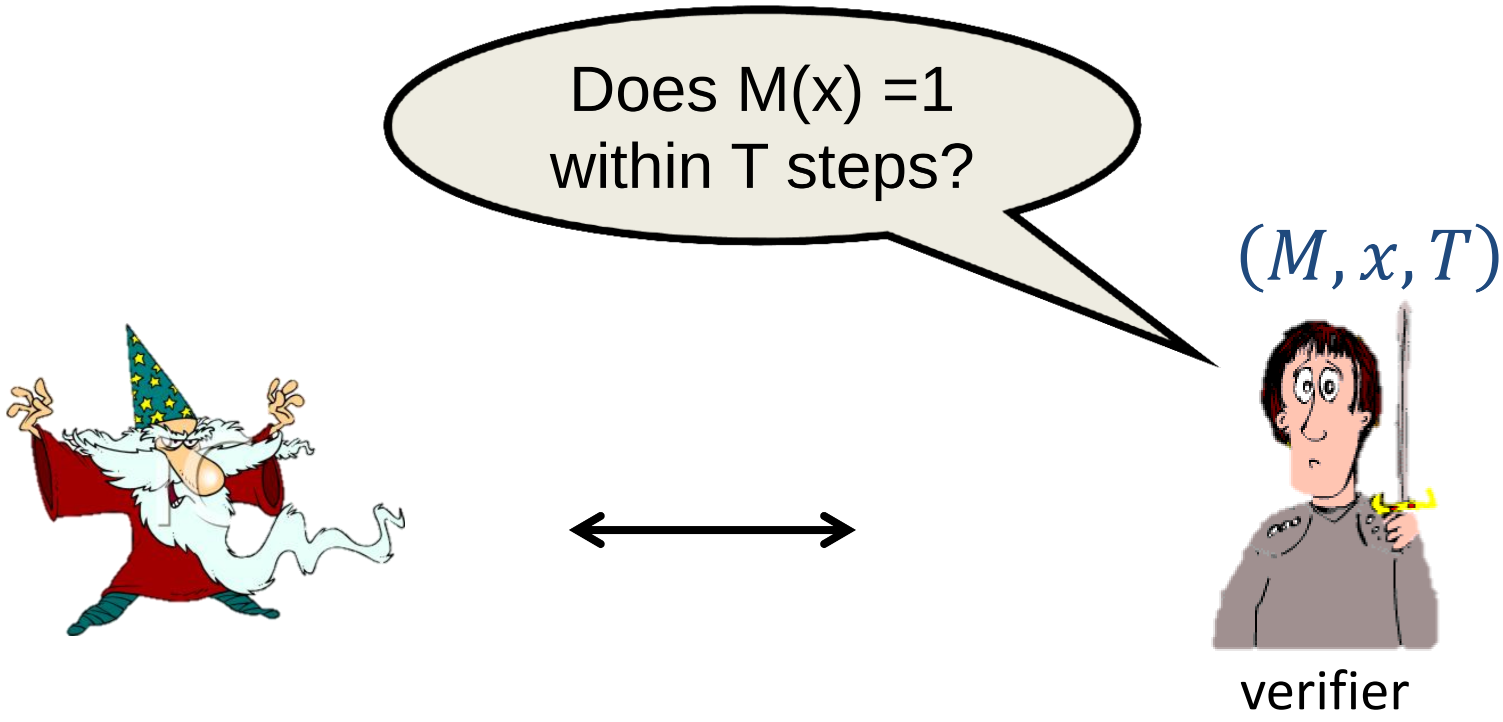
How quickly can we verify the result of long computations?
(Plain version)



Verify by running M(x) for T steps.

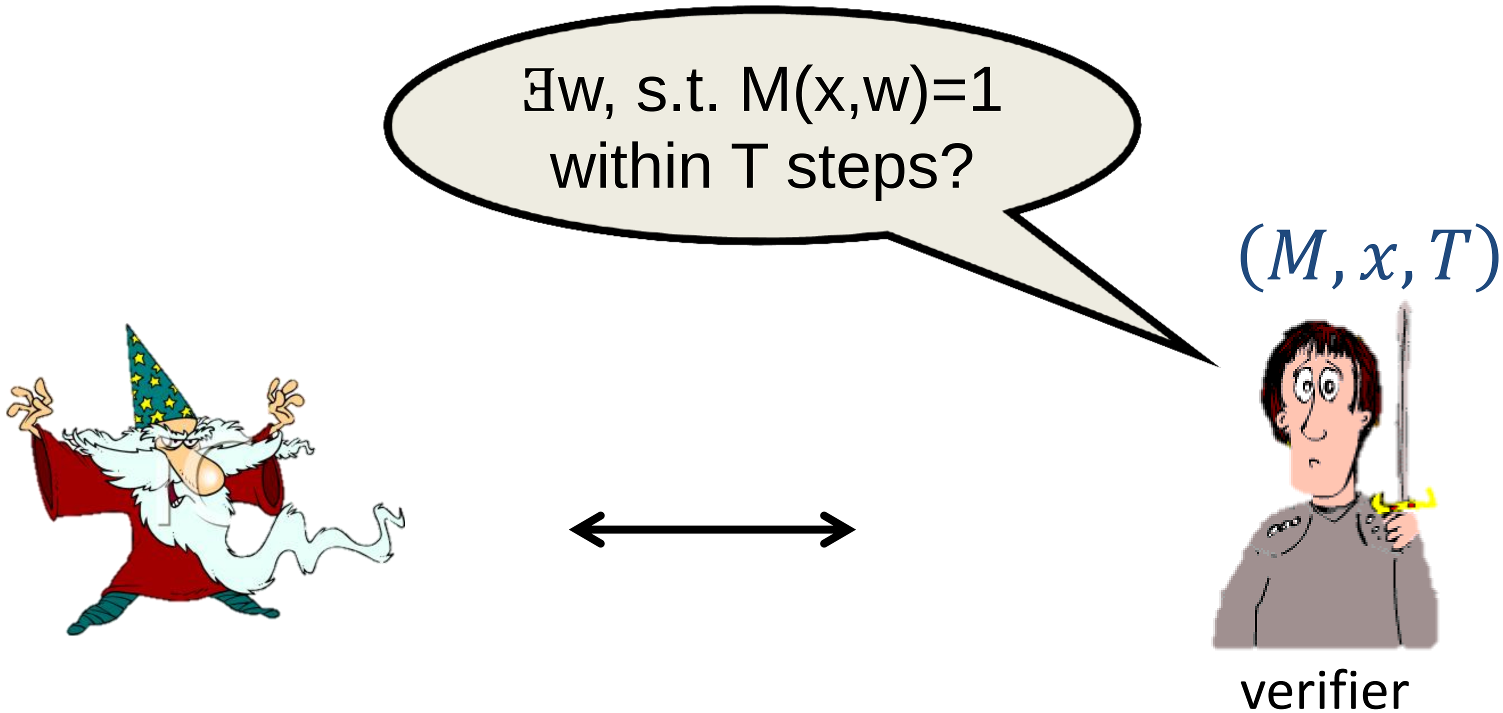
How quickly can we verify the result of long computations?

(Plain version)



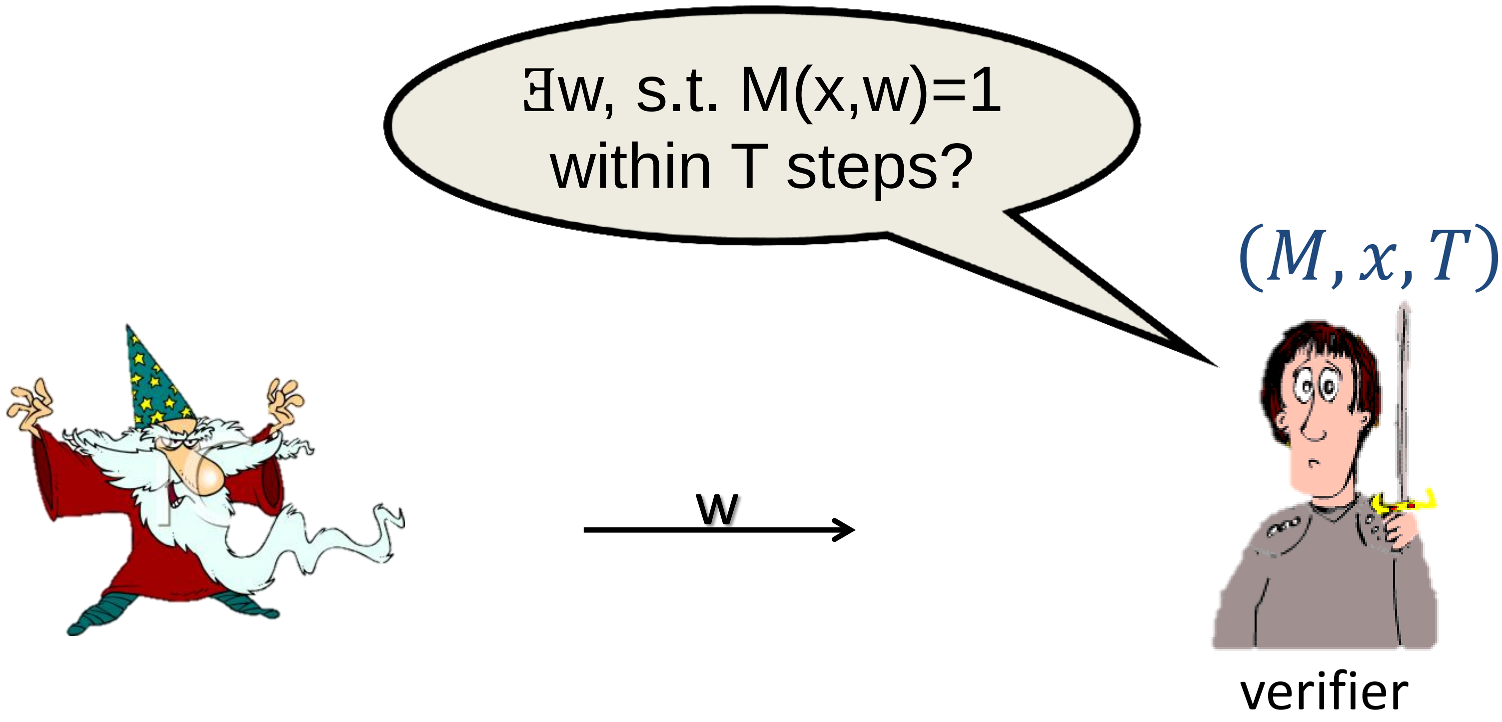
Can we do better?

How quickly can we verify the result of long computations?
(with prover input – “NP version”)



How quickly can we verify the result of long computations?

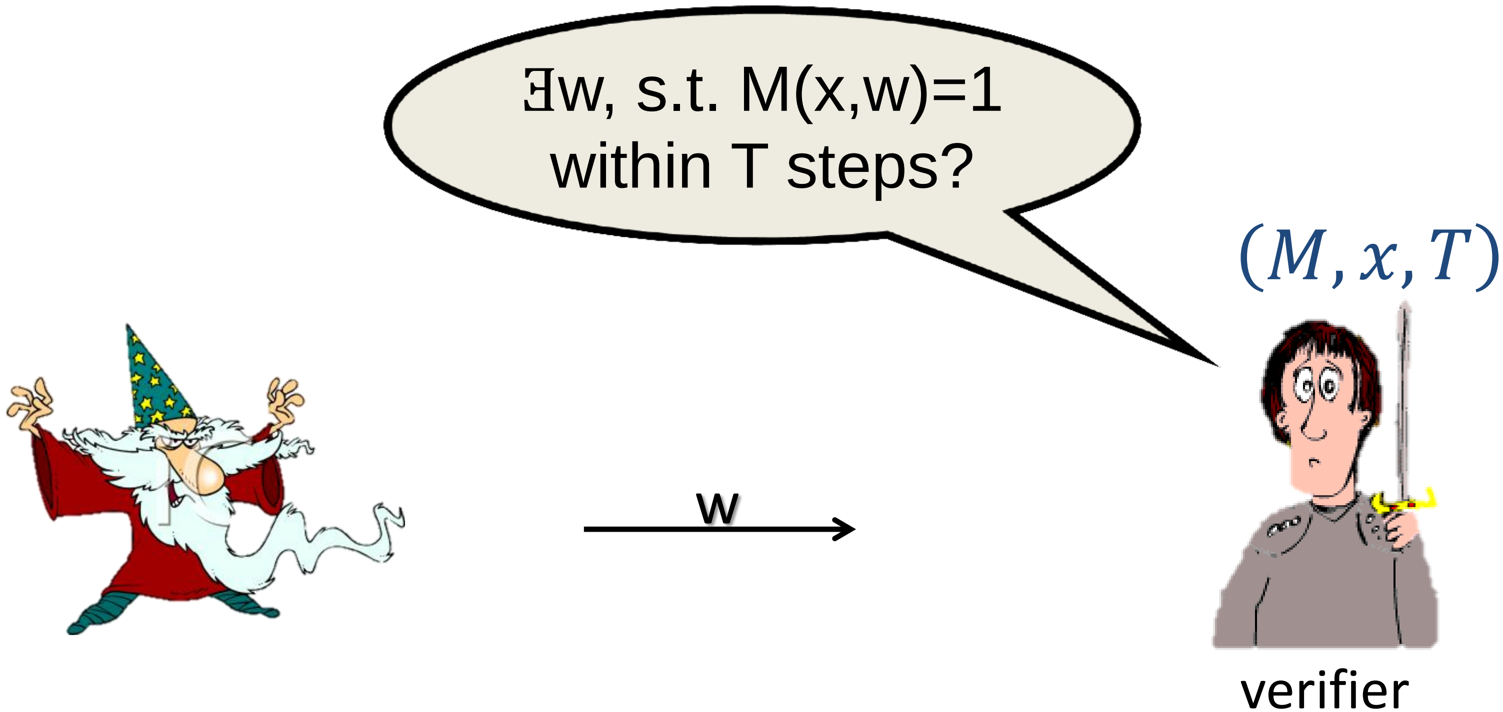
(with prover input – "NP version")



Verify by running M(x,w) for T steps.

How quickly can we verify the result of long computations?

(with prover input – “NP version”)



Can we do better?

Succinct Proofs with incomplete input
(“ for NP ”)

possibly long

$(M, x, T), w$

$\exists w$ s.t. $M(x, w) = 1$

in $\leq T$ steps?

↔

(M, x, T)

poly($|x|, k$)

universal poly, e.g.

$|x| \cdot k$

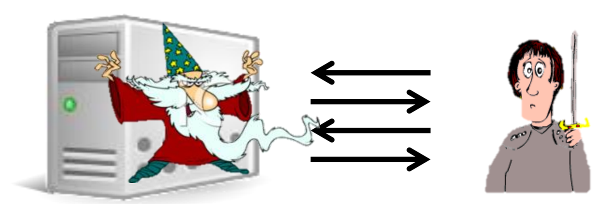
independent of T!

security parameter

Succinct Proofs with incomplete input
(“ for NP ”)

- Statistical soundness is unlikely [BHZ87, GH98, GVW02]. Thus we settle for computational soundness.

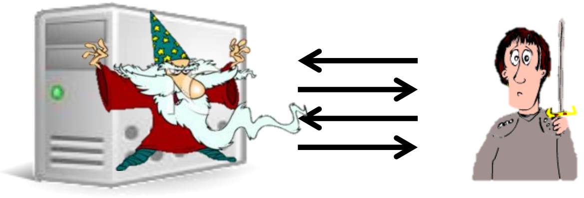
- However, we require extractability:
  - Natural in real-life applications (databases…)
  - Crucial for this work



How many rounds do succinct arguments require?

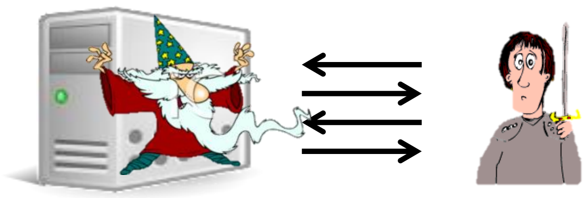
How many rounds do succinct arguments require?

[Kilian 92]: can do 4-message (assuming CRH)

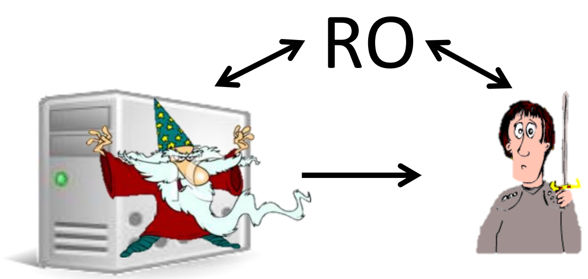


How many rounds do succinct arguments require?

[Kilian 92]: can do 4-message
(assuming CRH)



[Micali 94]: one message!
with a random oracle
(aka “CS proofs”)

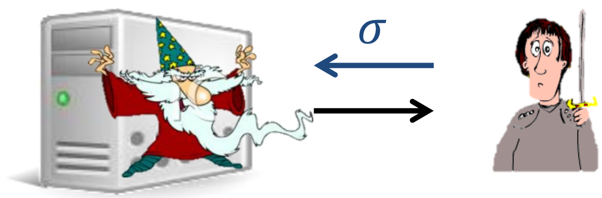


Non-interactive in the plain model?

Non-interactive in the plain model?

Totally non-interactive protocols
(against non-uniform provers
for “hard enough languages”)
Are unlikely [BHZ87, GH98, GVW02].

With a verifier initial message (reference string)?



reference string σ
sent before statements

Succinct Non-Interactive Argument of Knowledge (SNARK):

A protocol (P,V) such that:

- V sends an initial message σ to P
- Repeat: - P sends (M,x,T), π to V
  - V(M,x,T, π, σ)=acc/rej

Succinct Non-Interactive Argument of Knowledge (SNARK):

A protocol (P,V) such that:

- V sends an initial message σ to P
- Repeat: - P sends (M,x,T), π to V
  - V(M,x,T, π, σ)=acc/rej

Completeness: If ∃w s.t. M(x,w)=1 within T steps, then V accepts.

Extractability: ∀ pt P' ∃ pt E, such that when (P',V) accepts (M,x,t, π), E outputs w s.t. M(x,w)=1 within T steps (except w.p. negl(k)).

Designated verifier SNARKs

Same as (publicly verifier) SNARKs except:

- V keeps secret state τ associated with σ .
- V uses τ in each verification.

Disadvantages:

- Vulnerable to leakage on verifier (even the verifier's decision)
- Proofs are no longer transferrable or publicly verifiable ("publishable").
- Harder to compose (later on)

Can we construct SNARKs?

No SNARK can be proven secure via “black-box reduction to an efficiently falsifiable assumption”

[Gentry-Wichs11].

- even for designated verifier SNARKs
- even if we only require plain soundness (without knowledge extraction)

Can we construct SNARKs?

No SNARK can be proven secure via "black-box reduction to an efficiently falsifiable assumption"
[Gentry-Wichs11].

- even for designated verifier SNARKs
- even if we only require plain soundness (without knowledge extraction)

What can we do?

- Option 1: Use non BB reductions
- Option 2: Use other assumptions

SNARKs from "non--falsifiable assumptions"

- Replace the RO in [Micali94] with a "sufficiently complicated" hash function and assume security.

Disadvantages: Implementation specific, doesn't teach us much…

- Based on "extractable collision resistant hash functions" [Bitansky Canetti Chiesa Tromer 11 , Goldwasser Lin Rubinstein 11, Damgard Faust Hazay 11]

Disadvantage: Only designated verifier.

PV SNARKs with long reference string ("with pre-processing")

In the initial stage, V "works hard":

generates (σ,τ) where:

- τ is poly(k)
- σ is poly(T,k)

In proof stage, V is still succinct - only uses τ.

PV SNARKs with long reference string (“with pre-processing”)

In the initial stage, V “works hard”:

generates (σ,τ) where:

- τ is poly(k)
- σ is poly(T,k)

In proof stage, V is still succinct - only uses τ.

Note: τ is public!

Can realize based on a Knowledge-of-exponent assumption in bilinear groups

[Groth10, Lipmaa12, Gennaro-Gentry-Parno-Raykova12]

Another advantage of [G10,L12,GGPR12]
(Following [Ishai-Kushilevitz-Ostrovsky07])

Very different techniques – alternative to PCPs

Potentially better efficiency (for prover).

Prover efficiency is important !
(e.g. cloud computing)

Another advantage of [G10,L12,GGPR12]
(Following [Ishai-Kushilevitz-Ostrovsky07])

Very different techniques – alternative to PCPs

Potentially better efficiency (for prover).

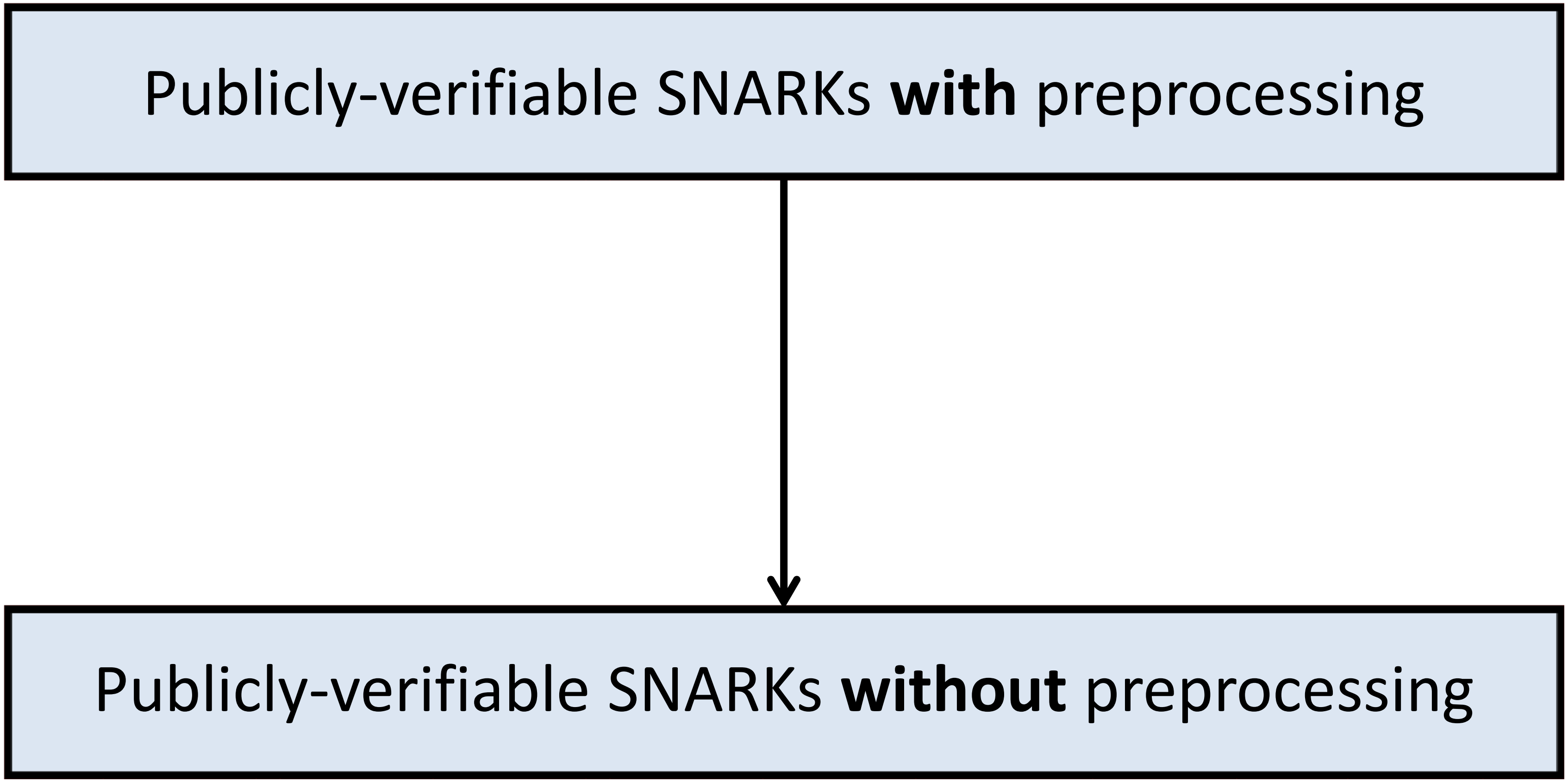
But…

For computations with time T , space S

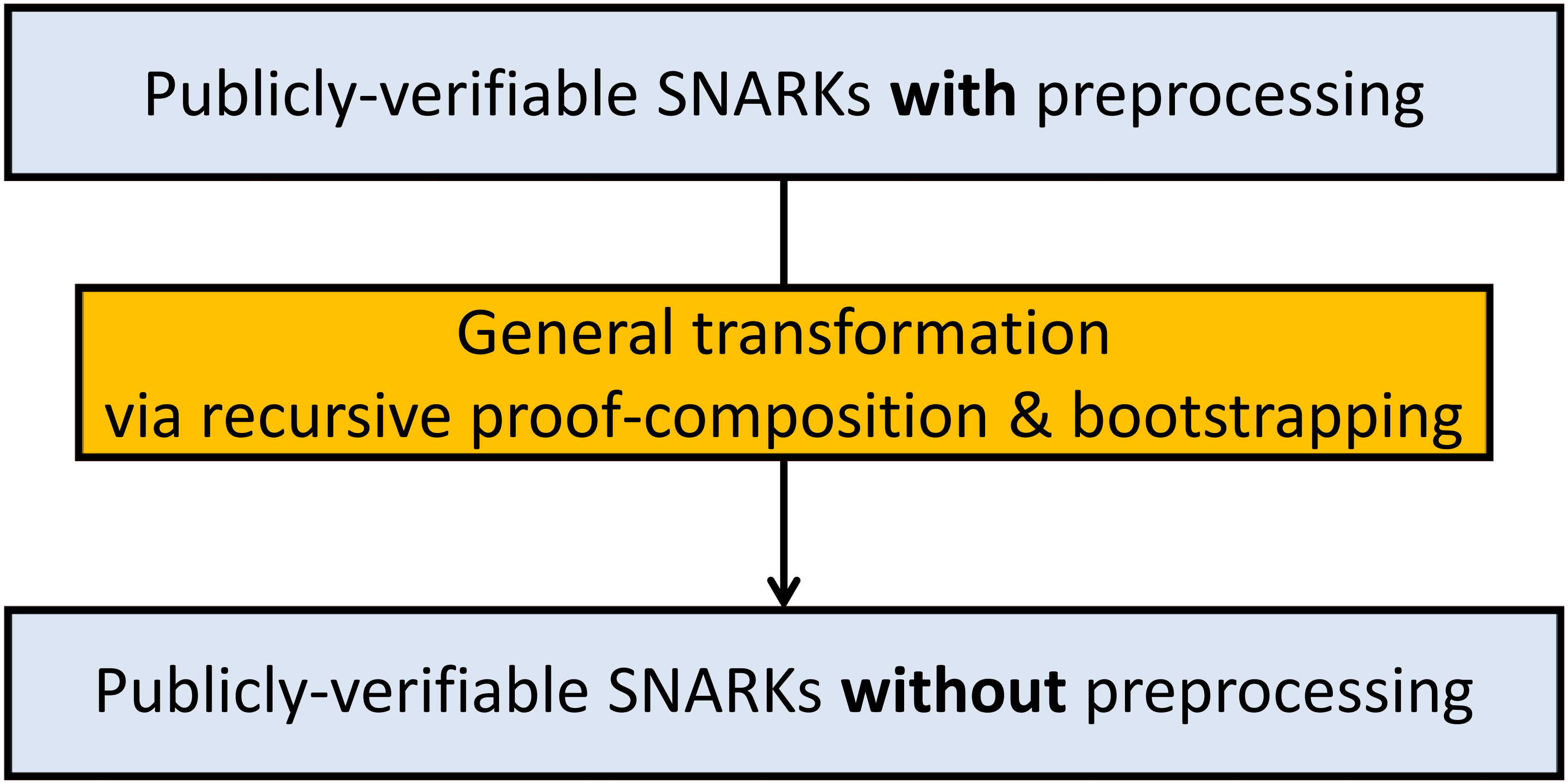
Prover needs T poly(k) ***space!***

Would like to preserve time and space individually.

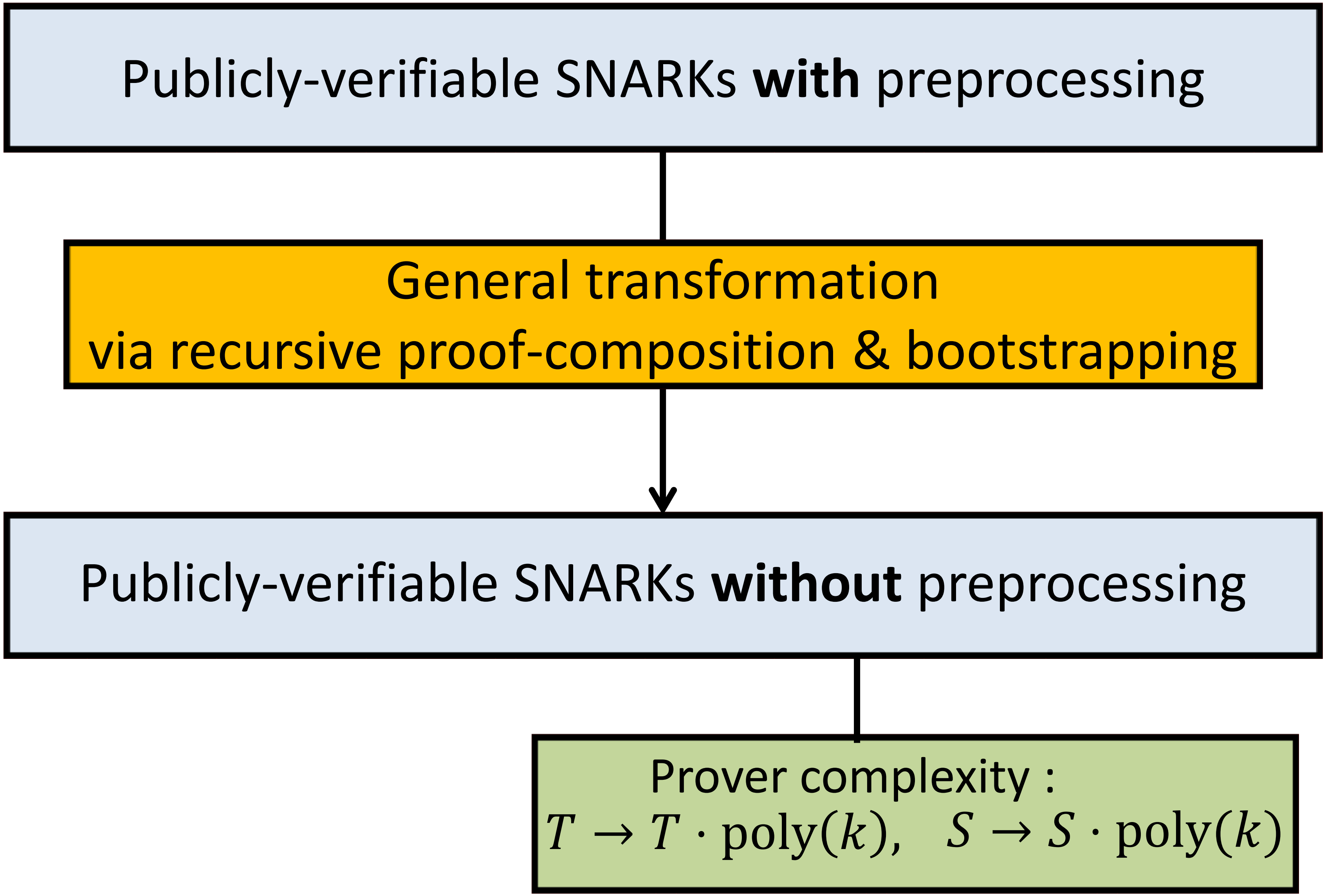
First Main Result



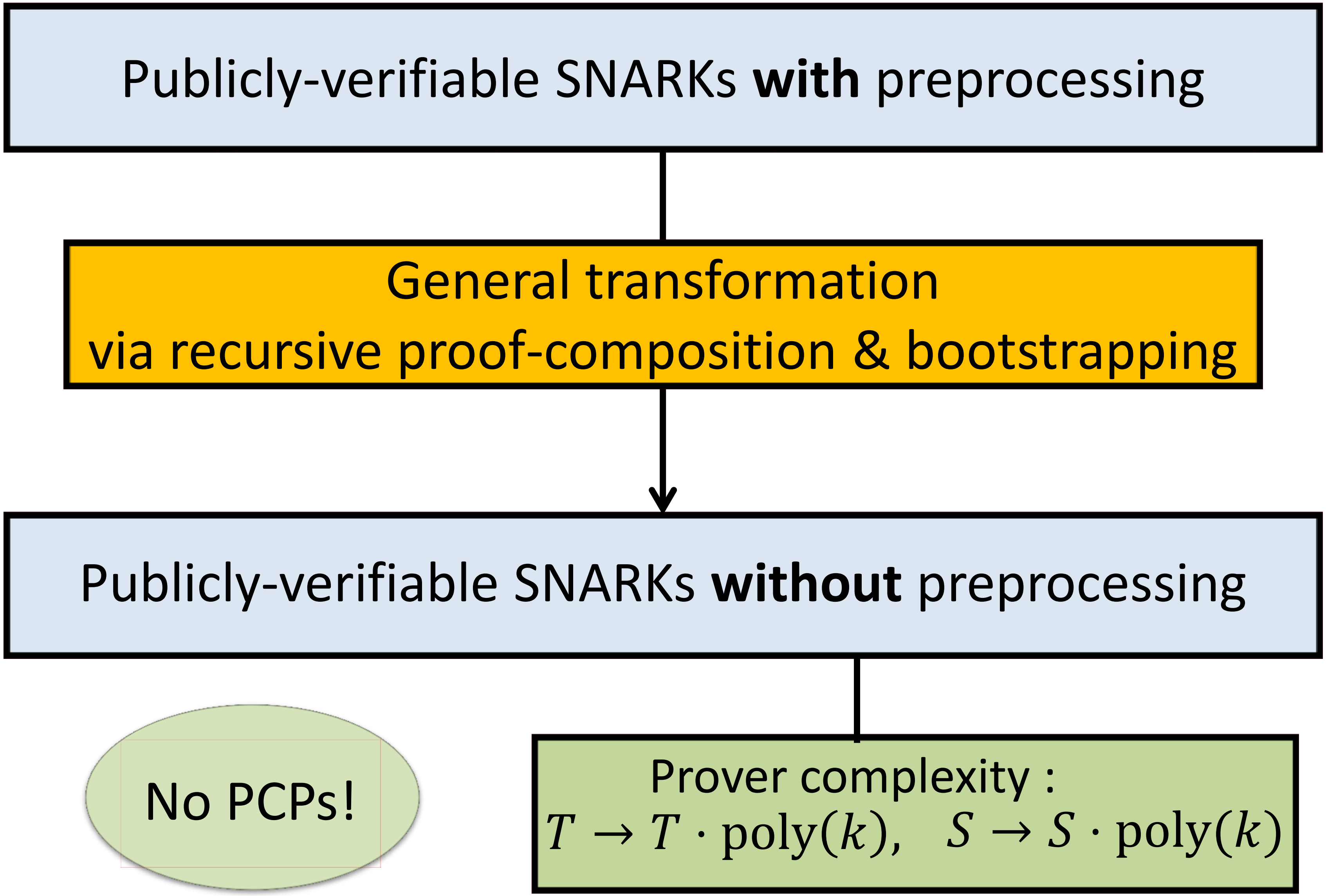
First Main Result



First Main Result



First Main Result



Corollaries

Assuming KEA in a bilinear group, there exist

fully succinct publicly-verifiable SNARKs .

Any SNARK can be transformed into a SNARK where:

- Prover time is $T \cdot poly(k)$

- Prover space is $S \cdot poly(k)$

(T,S are time and space of original M)

The Core Idea:
Bootstrapping a SNARK

Only need to be able to prove correctness of (many) small computations.

The Core Idea: Bootstrapping a SNARK

Only need to be able to prove correctness of (many) small computations.

How small?

as small as SNARK verification (and a bit more)

- ➔ The preprocessing becomes cheap (poly(k))
- ➔ Prover overhead becomes poly(k) (both in time and in space)

Part I

How to Bootstrap a SNARK:
a Bare-Bones Description

Part II

Using the Proof Carrying Data
(PCD) abstraction

Part I:
Bare-Bones Description

Incremental Computation [Valiant08]
a possibly useful idea

Compile a computation $M(x, w)$ to a new one that after each step spits a **short** proof of its correctness **so far**

Incremental Computation [Valiant08]
a possibly useful idea

Compile a computation $\boldsymbol{M(x,w)}$ to a new one that after each step spits a **short** proof of its correctness **so far**

but… (implicitly) assumes fully-succinct SNARKs

Incremental Computation [Valiant08]
a possibly useful idea

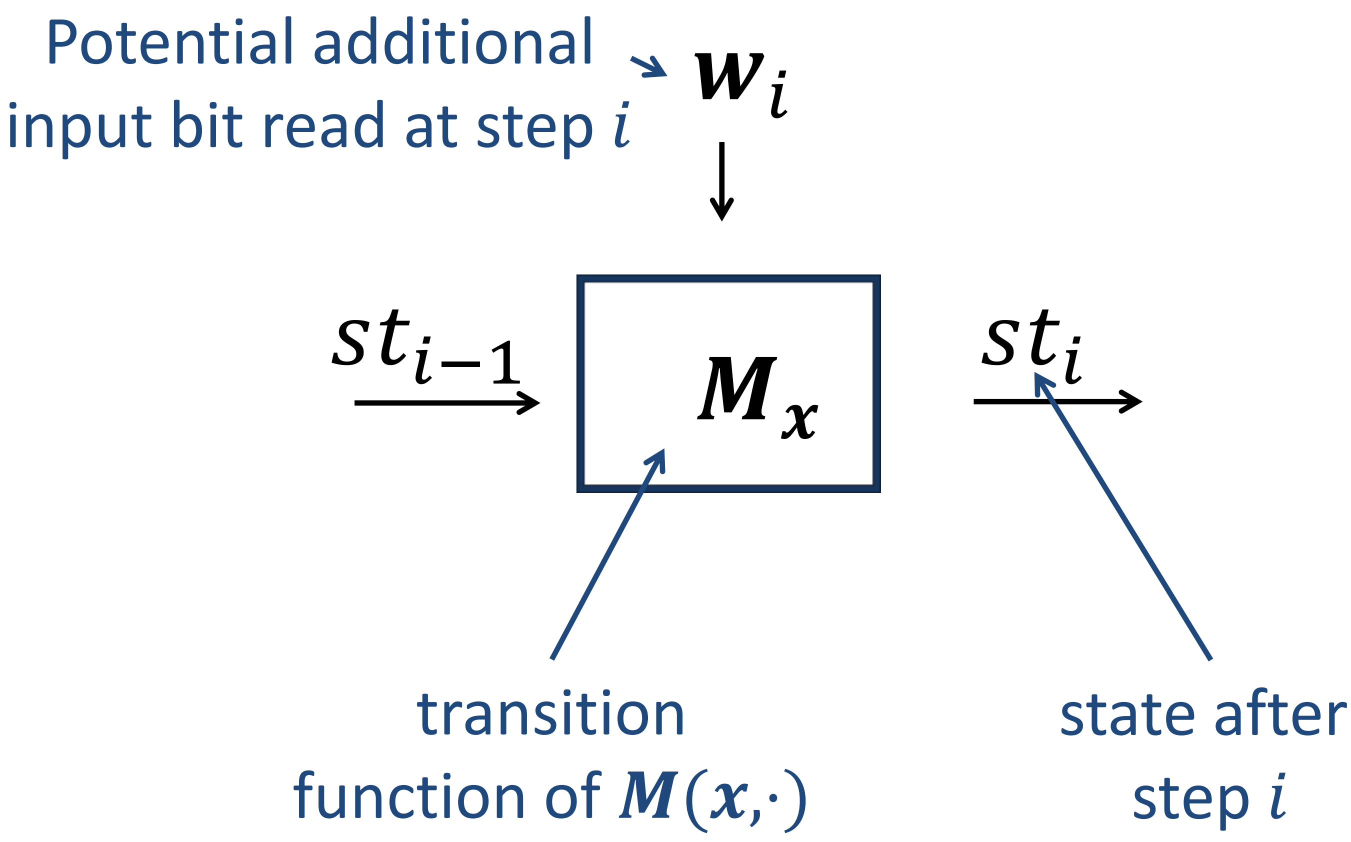
Still uses SNARKs in a non-trivial way:
proofs only involve "small" computations:
proportional to the space S used by $\boldsymbol{M}$.

Can use preprocessing SNARKs, where
preprocessing is as cheap as S ….

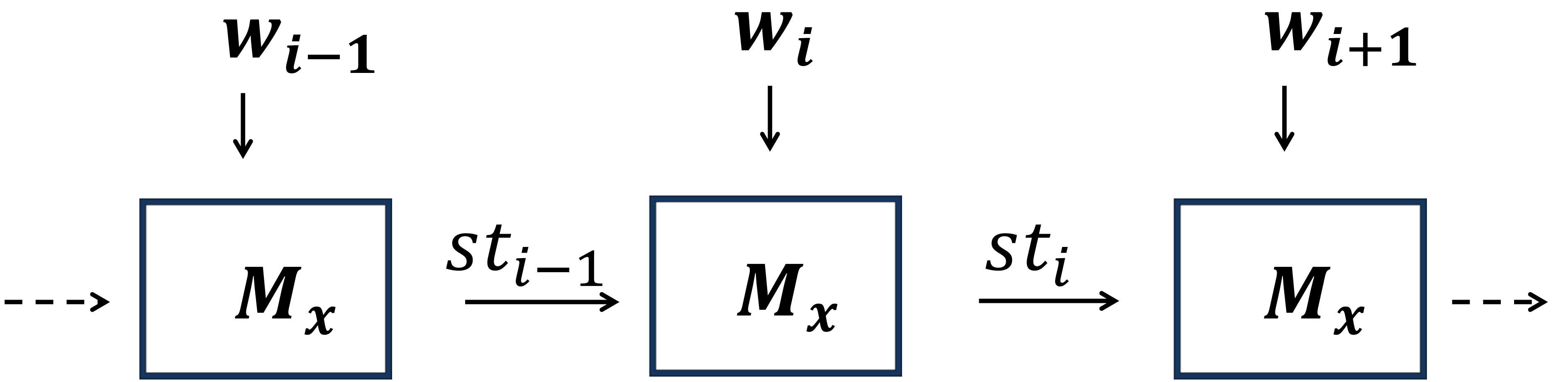
Problems:
In general, S may be as large as T
Need to carefully aggregate proofs by composition

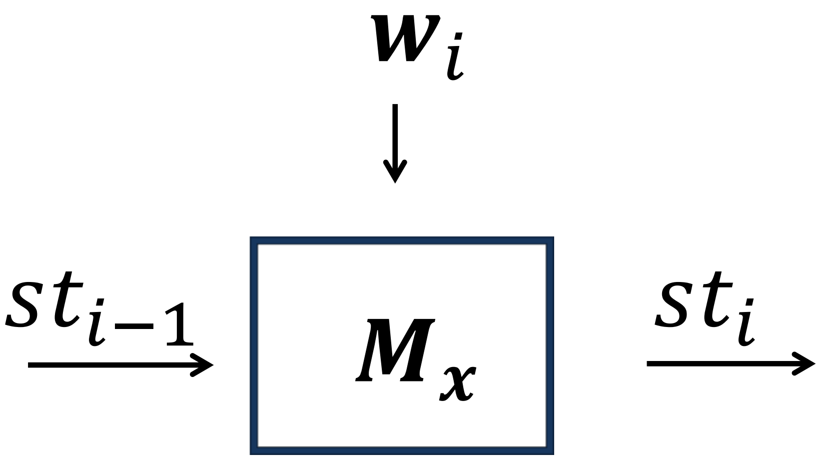
Incremental Computation More Concretely

Split a T-step computation $\boldsymbol{M}(\boldsymbol{x}, \boldsymbol{w})$ to T single-step computations



Split a T-step computation $\boldsymbol{M}(\boldsymbol{x}, \boldsymbol{w})$ to T single-step computations

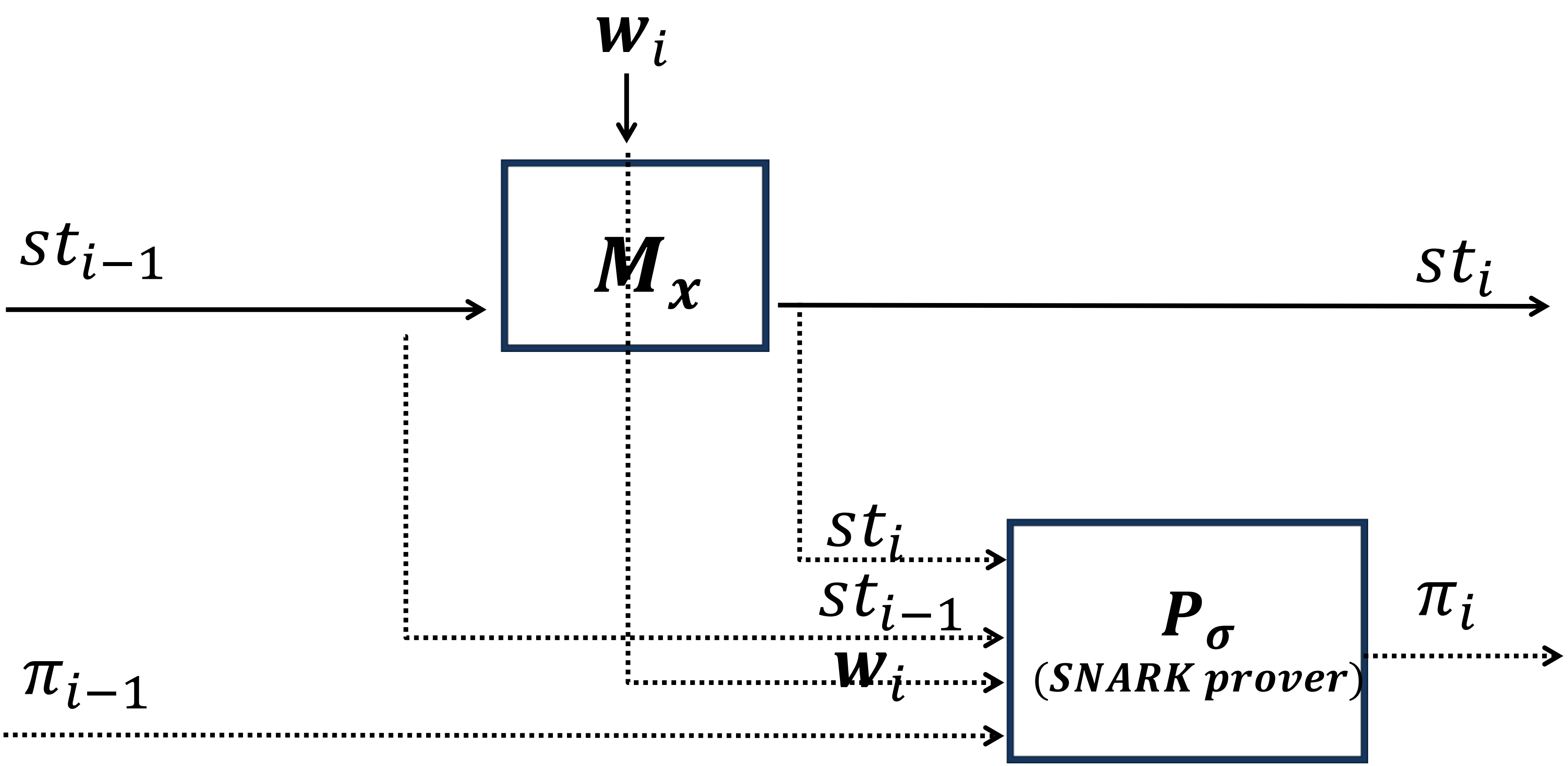




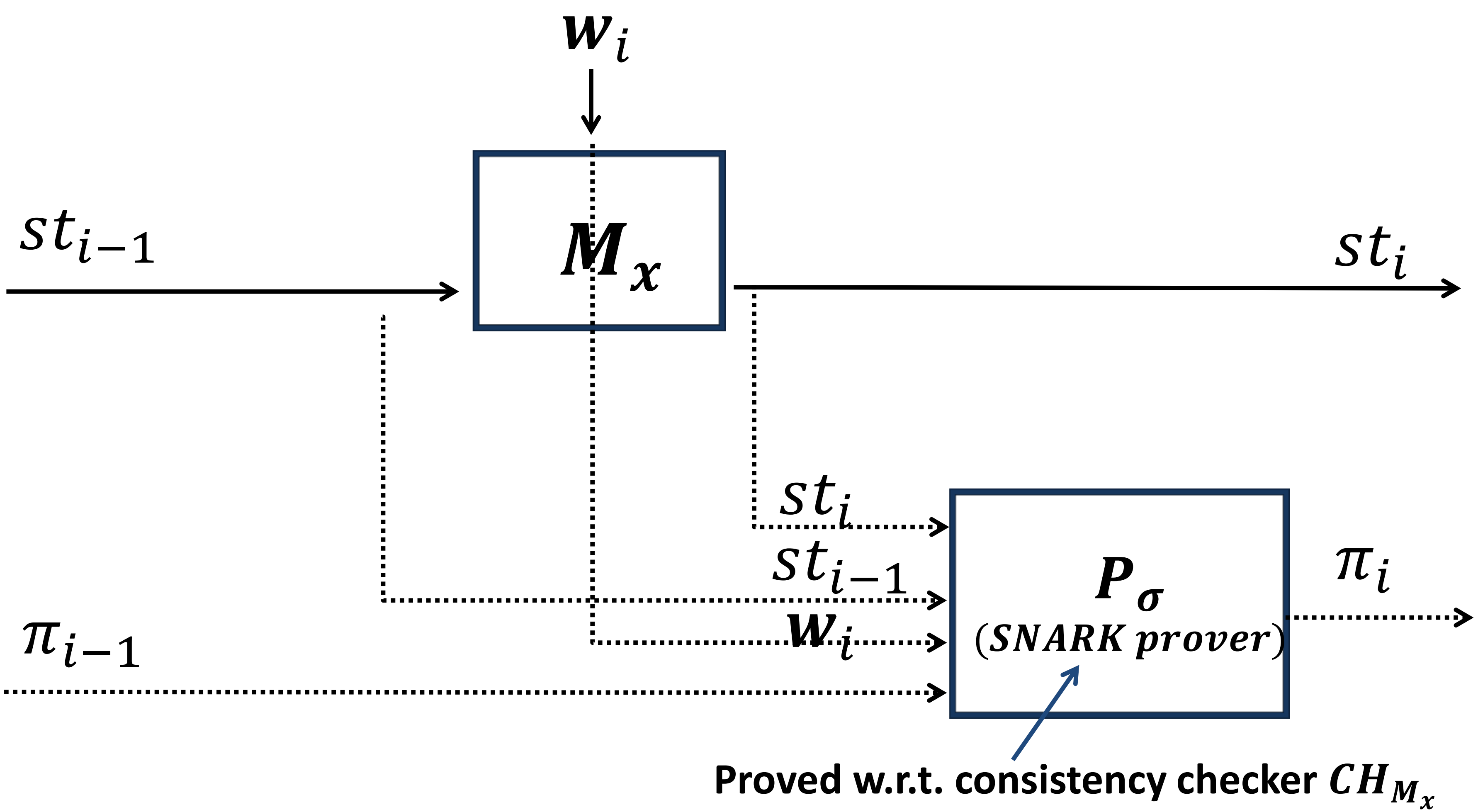
Compose short proof for current step with short proof for previous steps:

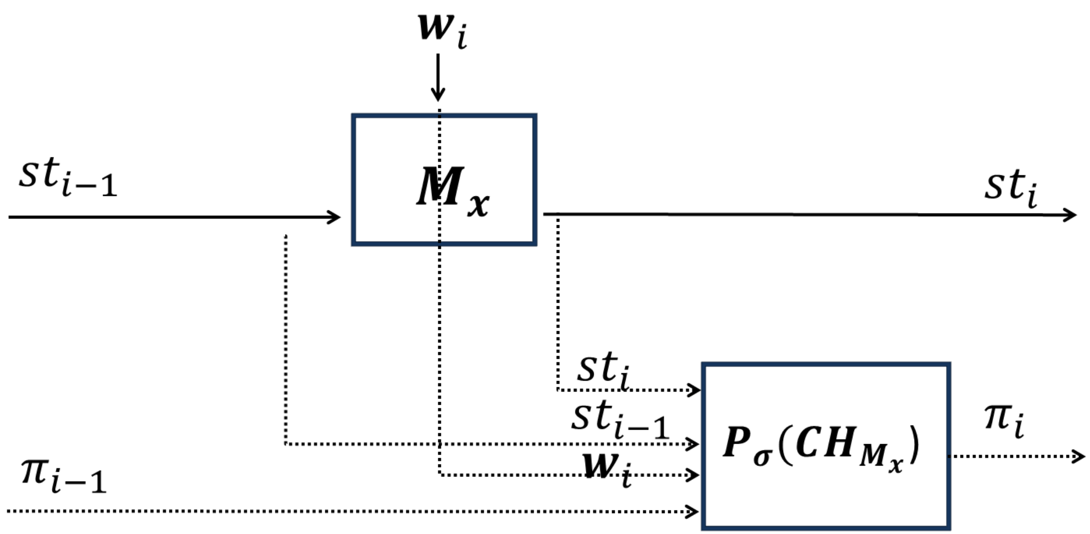
1. performed step i correctly
2. verified a proof π_{i-1} for correctness of steps $1 \ldots i-1$

Augment computation $\boldsymbol{M(x,w)}$ with consistency proofs



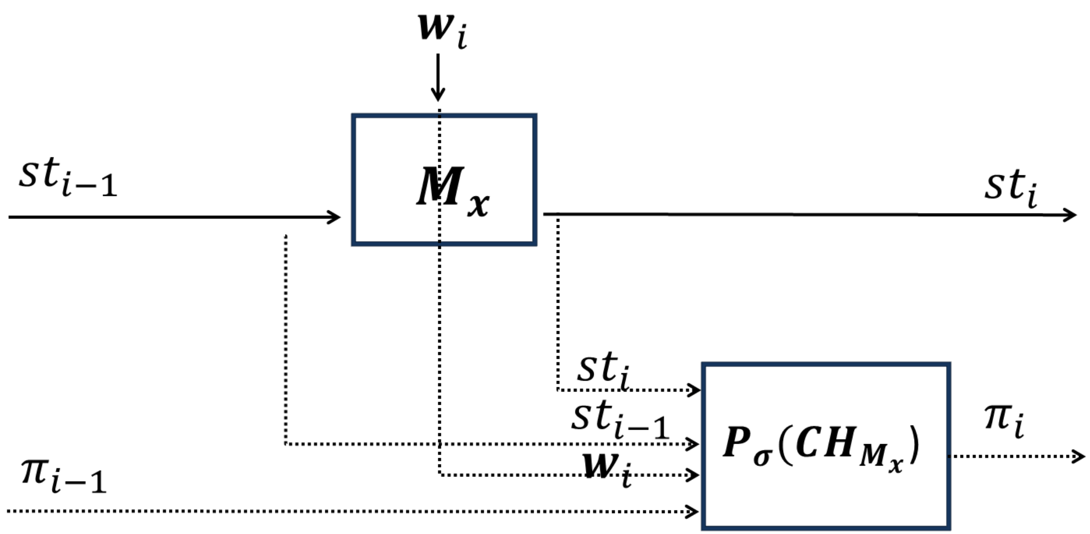
Augment computation $\boldsymbol{M(x, w)}$ with consistency proofs





Recursive Consistency Checker $\boldsymbol{CH_{M_x}}$

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$



Recursive Consistency Checker $\boldsymbol{CH_{M_x}}$

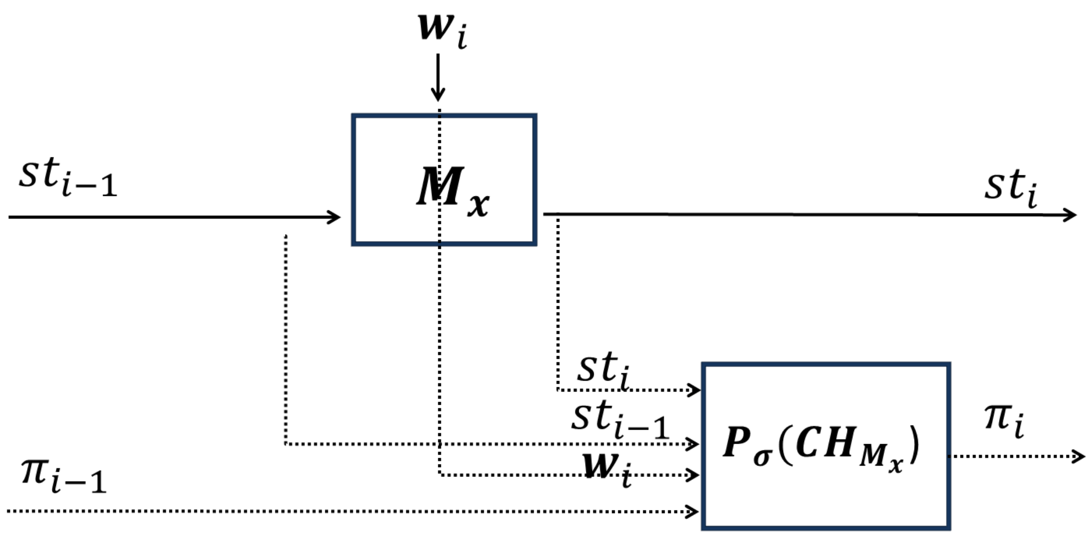
Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$

If $i = 0$ and $st_0 = initial\ state,$ accept.

else check:

$\boldsymbol{M_x}(st_{i-1}; \boldsymbol{w_i}) = st_i$

$V_\tau(\boldsymbol{CH_{M_x}}, st_{i-1}, \pi_{i-1})$=acc



Recursive Consistency Checker CH_{M_x}

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, w_i)$

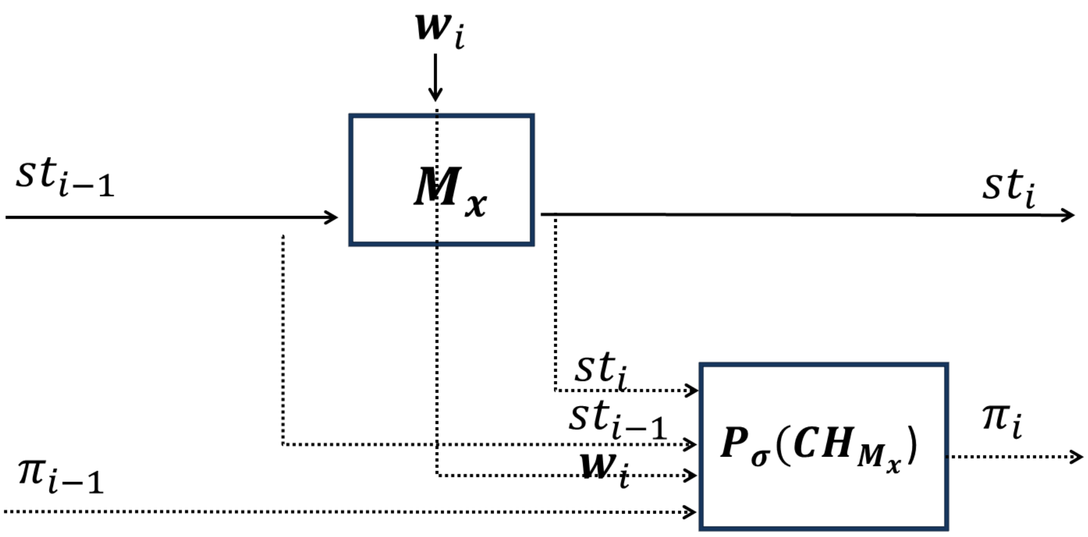
If $i = 0$ and $st_0 = initial\ state,$ accept.

else check:

$M_x(st_{i-1}; w_i) = st_i$

$V_\tau(CH_{M_x}, st_{i-1}, \pi_{i-1})$=acc

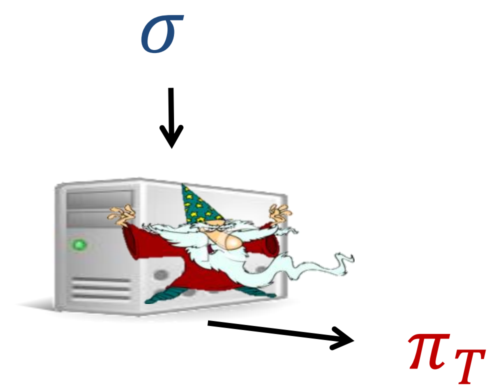
need recursion

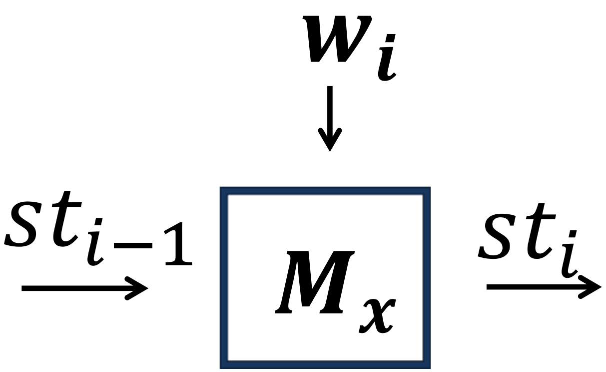


Is the resulting proof sound?

$$V_\tau(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_T; \pi_T) = 1$$

$$st_T = \text{“accept”}$$





Recursive Consistency Checker $\boldsymbol{CH}_{\boldsymbol{M_x}}$

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\boldsymbol{M_x}$ accept.

else check:

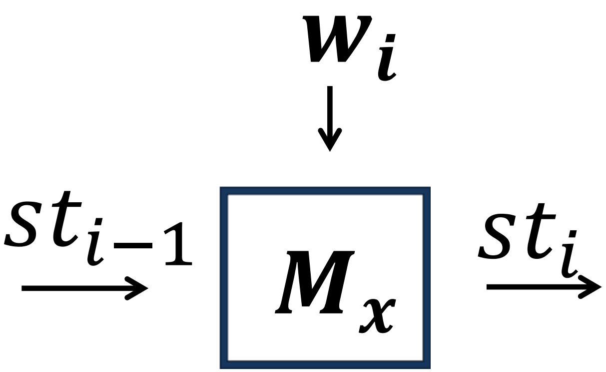
$\boldsymbol{M_x}(st_{i-1}; \boldsymbol{w_i}) = st_i$

V_τ accepts π_{i-1} for statement “ $\boldsymbol{CH}_{\boldsymbol{M_x}}$ accepts st_{i-1}

$$\exists(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T}):$$
$$\boldsymbol{M_x}(st_{T-1}, \boldsymbol{w_T}) = st_T$$
$$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-1}; \pi_{T-1}) = 1$$

$\longleftarrow$

$$V_\tau(\boldsymbol{CH_{M_x}}; st_T; \pi_T) = 1$$
$$st_T = \text{“accept”}$$



Recursive Consistency Checker $\boldsymbol{CH_{M_x}}$

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\boldsymbol{M_x}$ accept.
else check:
$\boldsymbol{M_x}(st_{i-1}; \boldsymbol{w_i}) = st_i$
V_τ accepts π_{i-1} for statement “ $\boldsymbol{CH_{M_x}}$ accepts st_{i-1}

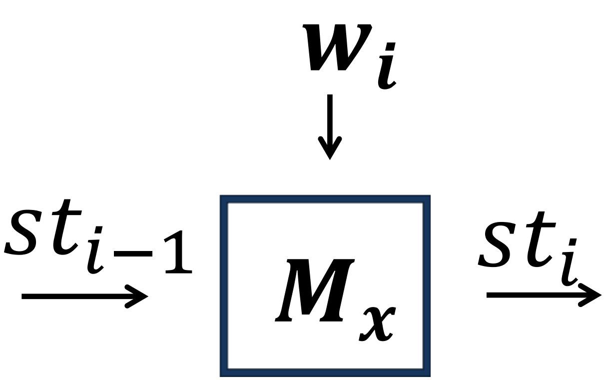
$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_T; \pi_T\right) = 1$$
$$st_T = \text{“accept”}$$

$\leftarrow$

$$\exists(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T}):$$
$$\boldsymbol{M_x}(st_{T-1}, \boldsymbol{w_T}) = st_T$$
$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_{T-1}; \pi_{T-1}\right) = 1$$

$\downarrow$

$$\exists(\pi_{T-2}, st_{T-2}, \boldsymbol{w_{T-1}}):$$
$$M_x(st_{T-2}, \boldsymbol{w_{T-1}}) = st_{T-1}$$
$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_{T-2}; \pi_{T-2}\right) = 1$$



Recursive Consistency Checker $\boldsymbol{CH}_{\boldsymbol{M_x}}$

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\boldsymbol{M_x}$ accept.

else check:

$\boldsymbol{M_x}(st_{i-1}; \boldsymbol{w_i}) = st_i$

V_τ accepts π_{i-1} for statement “ $\boldsymbol{CH}_{\boldsymbol{M_x}}$ accepts st_{i-1}

$$V_\tau(\boldsymbol{CH_{M_x}}; st_T; \pi_T) = 1$$
$$st_T = \text{“accept”}$$

←

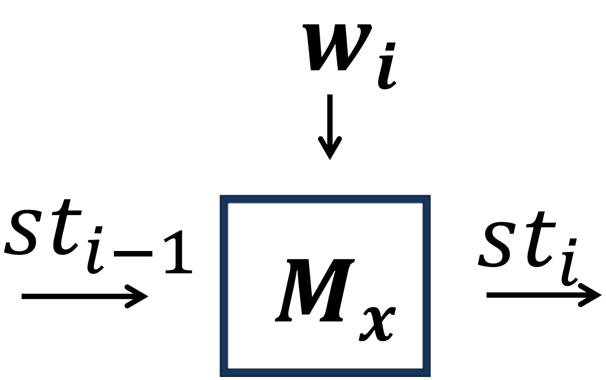
$$\exists(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T}):$$
$$\boldsymbol{M_x}(st_{T-1}, \boldsymbol{w_T}) = st_T$$
$$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-1}; \pi_{T-1}) = 1$$

↓

$$\exists(\pi_{T-2}, st_{T-2}, \boldsymbol{w_{T-1}}):$$
$$M_x(st_{T-2}, \boldsymbol{w_{T-1}}) = st_{T-1}$$
$$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-2}; \pi_{T-2}) = 1$$

- - - →

$$\exists(\pi_0, st_0, \boldsymbol{w_1}):$$
$$\boldsymbol{M_x}(st_0, \boldsymbol{w_1}) = st_1$$
$$st_0 = \text{“start”}$$



Recursive Consistency Checker $\boldsymbol{CH_{M_x}}$

Input: st_i witness: $(st_{i-1}, \pi_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\boldsymbol{M_x}$ accept.

else check:

$\boldsymbol{M_x}(st_{i-1}; \boldsymbol{w_i}) = st_i$

V_τ accepts π_{i-1} for statement “ $\boldsymbol{CH_{M_x}}$ accepts st_{i-1}

$$\exists(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T}):$$
$$\boldsymbol{M_x}(st_{T-1}, \boldsymbol{w_T}) = st_T$$
$$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-1}; \pi_{T-1}) = 1$$

$$V_\tau(\boldsymbol{CH_{M_x}}; st_T; \pi_T) = 1$$
$$st_T = \text{“accept”}$$

$$\exists(\pi_{T-2}, st_{T-2}, \boldsymbol{w_{T-1}}):$$
$$M_x(st_{T-2}, \boldsymbol{w_{T-1}}) = st_{T-1}$$
$$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-2}; \pi_{T-2}) = 1$$

$$\exists(\pi_0, st_0, \boldsymbol{w_1}):$$
$$\boldsymbol{M_x}(st_0, \boldsymbol{w_1}) = st_1$$
$$st_0 = \text{“start”}$$

➔ **Computational** soundness isn't enough

➔ Need knowledge extraction

➔ Need to apply the extraction recursively.

The extraction guarantee of SNARKs

$\forall$ prover P^*

ref string → σ

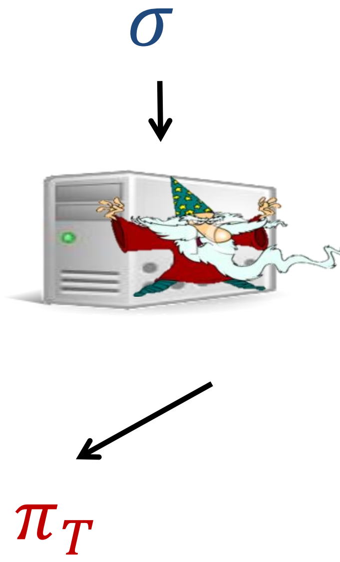
st_i, π_i that V_τ accepts

$\exists$ extractor E_{P^*}

σ

$wit = st_{i-1}, \pi_{i-1}, \boldsymbol{w_i}$

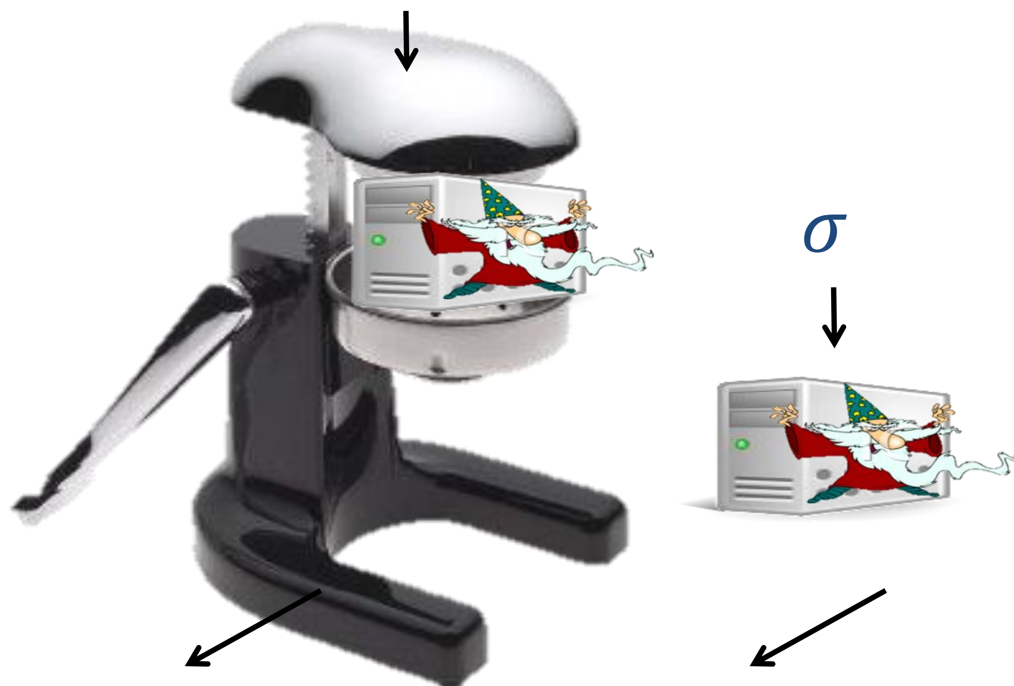
s.t. $\boldsymbol{CH_{M_x}(st_i, wit) = 1}$


σ
π_T

$$V_\tau(\boldsymbol{CH}_{\boldsymbol{M}_{\boldsymbol{x}}}; st_{T-1}; \pi_{T-1}) = 1$$

$$V_\tau(\boldsymbol{CH}_{\boldsymbol{M}_{\boldsymbol{x}}}; st_T; \pi_T) = 1$$

σ σ

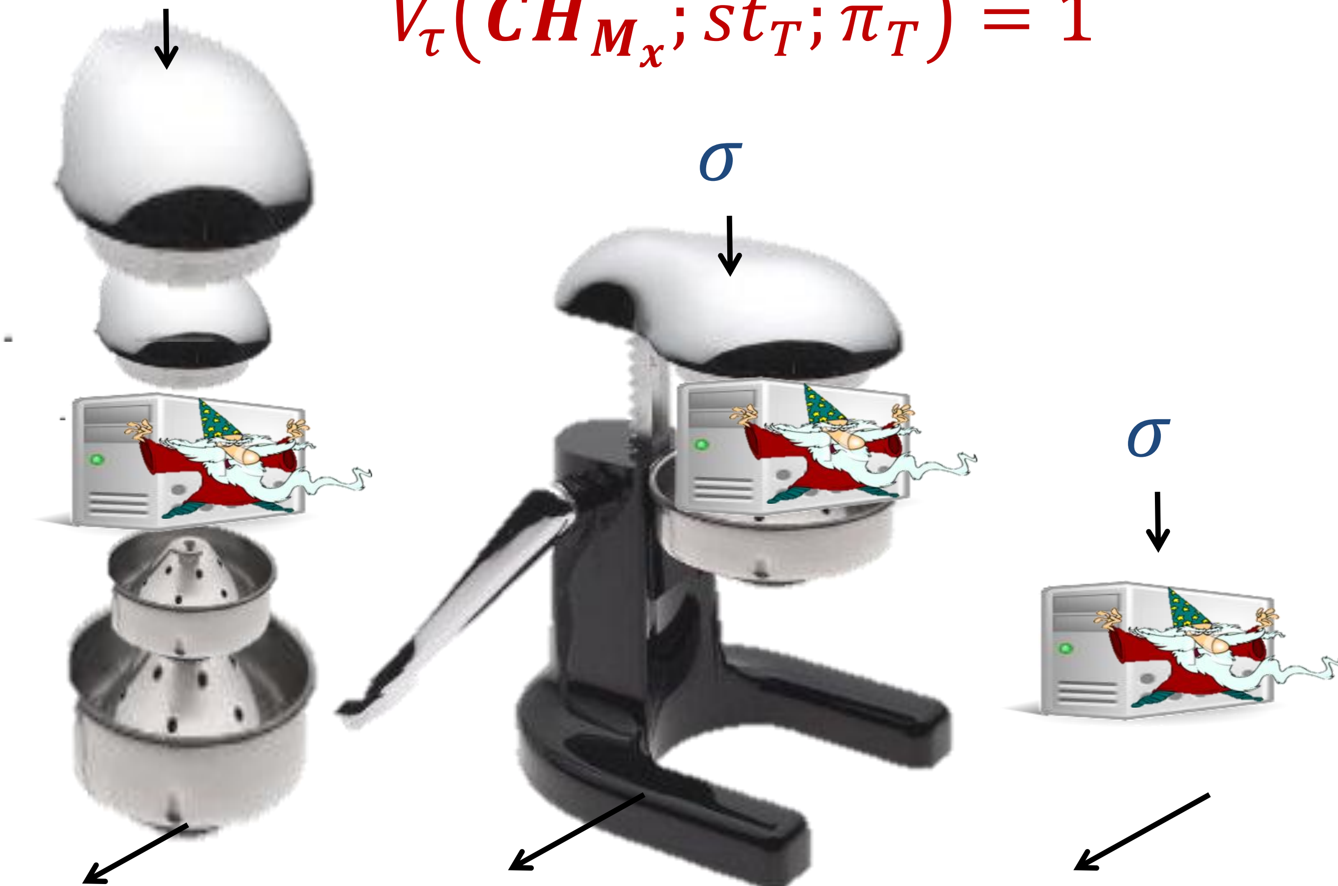


$(\pi_{T-1}, st_{T-1}, \boldsymbol{w}_{\boldsymbol{T}})$ π_T

$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M}_x}; st_{T-2}; \pi_{T-2}\right) = 1$$

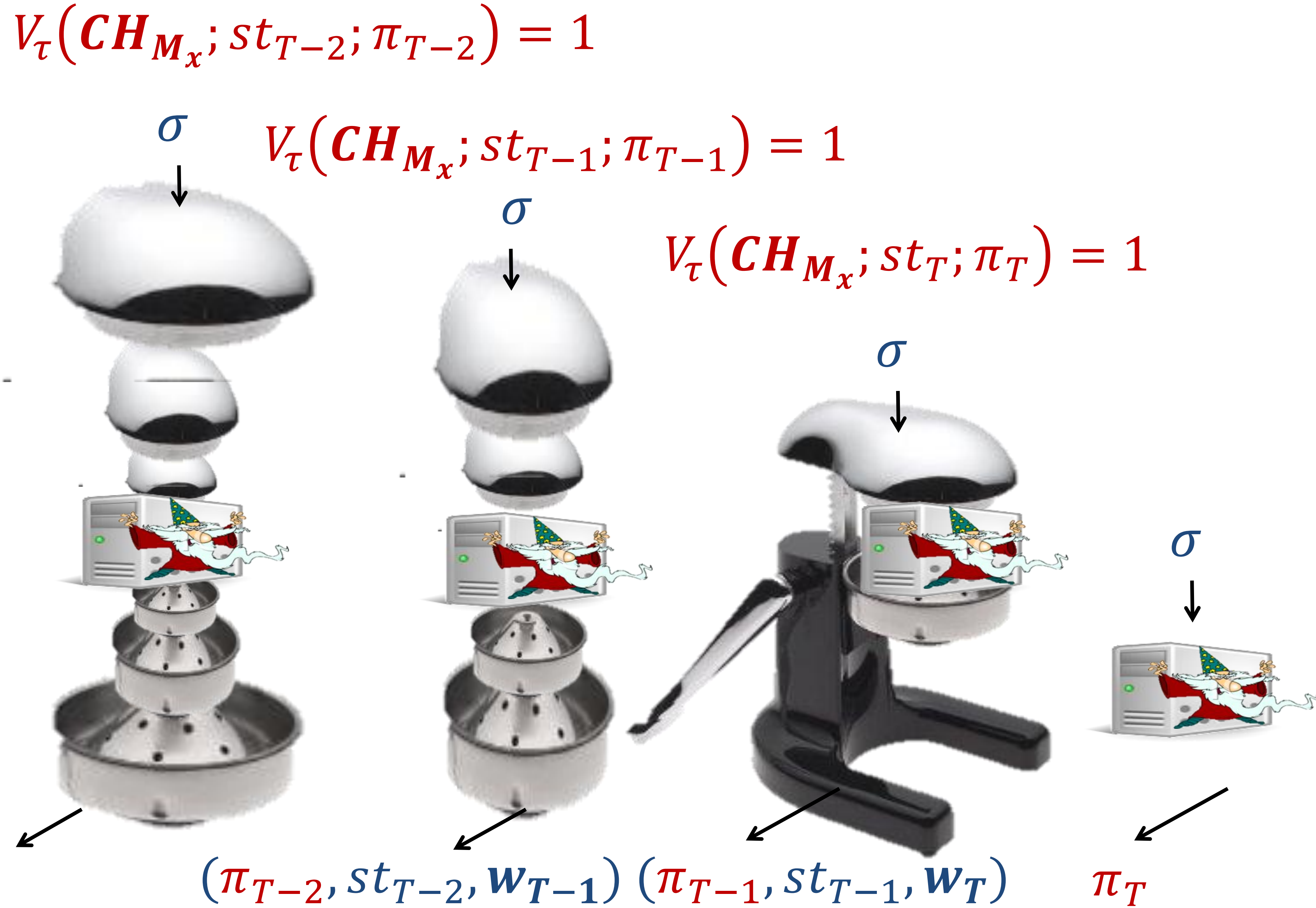
$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M}_x}; st_{T-1}; \pi_{T-1}\right) = 1$$

$$V_\tau\left(\boldsymbol{CH}_{\boldsymbol{M}_x}; st_T; \pi_T\right) = 1$$

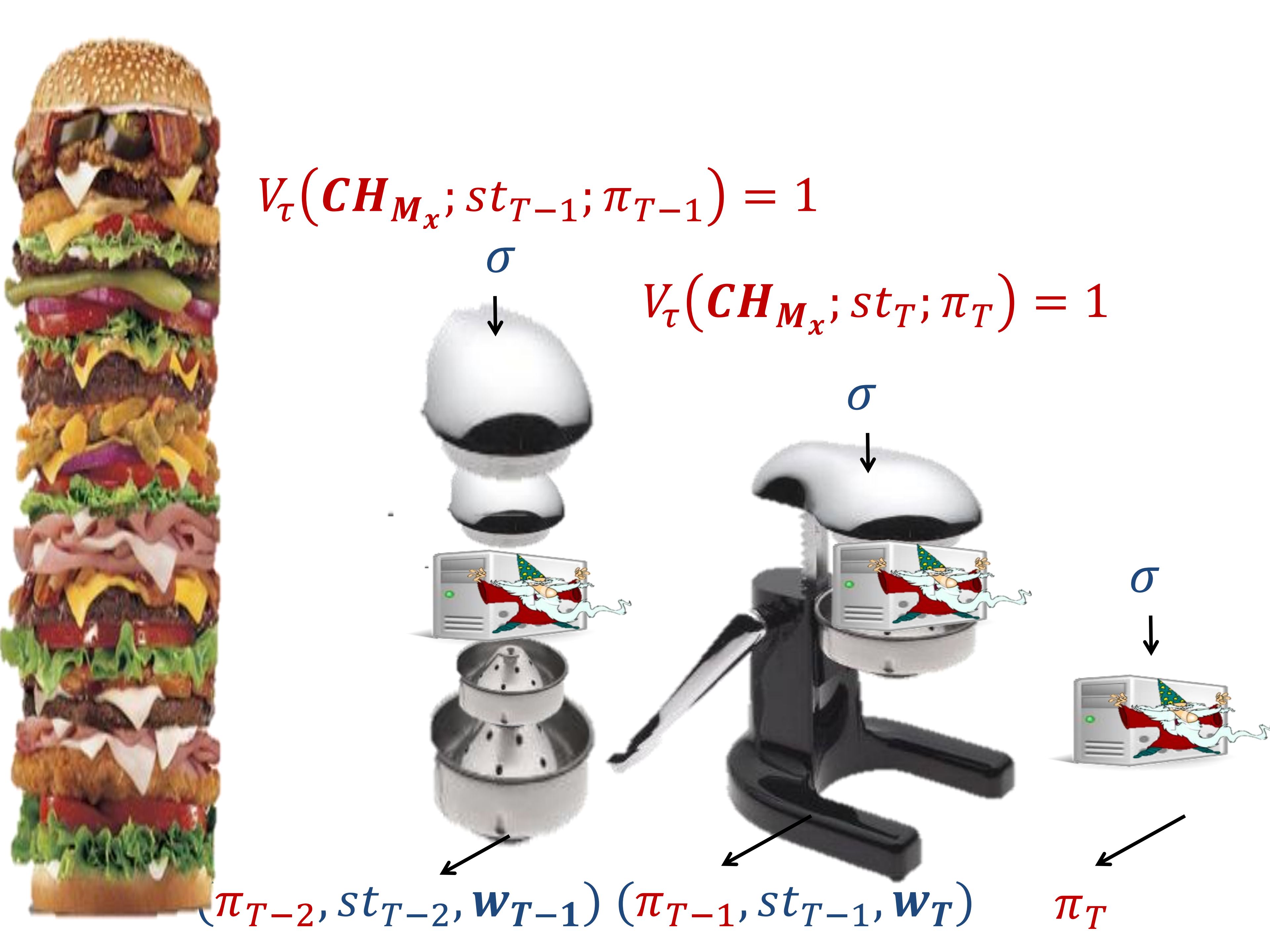


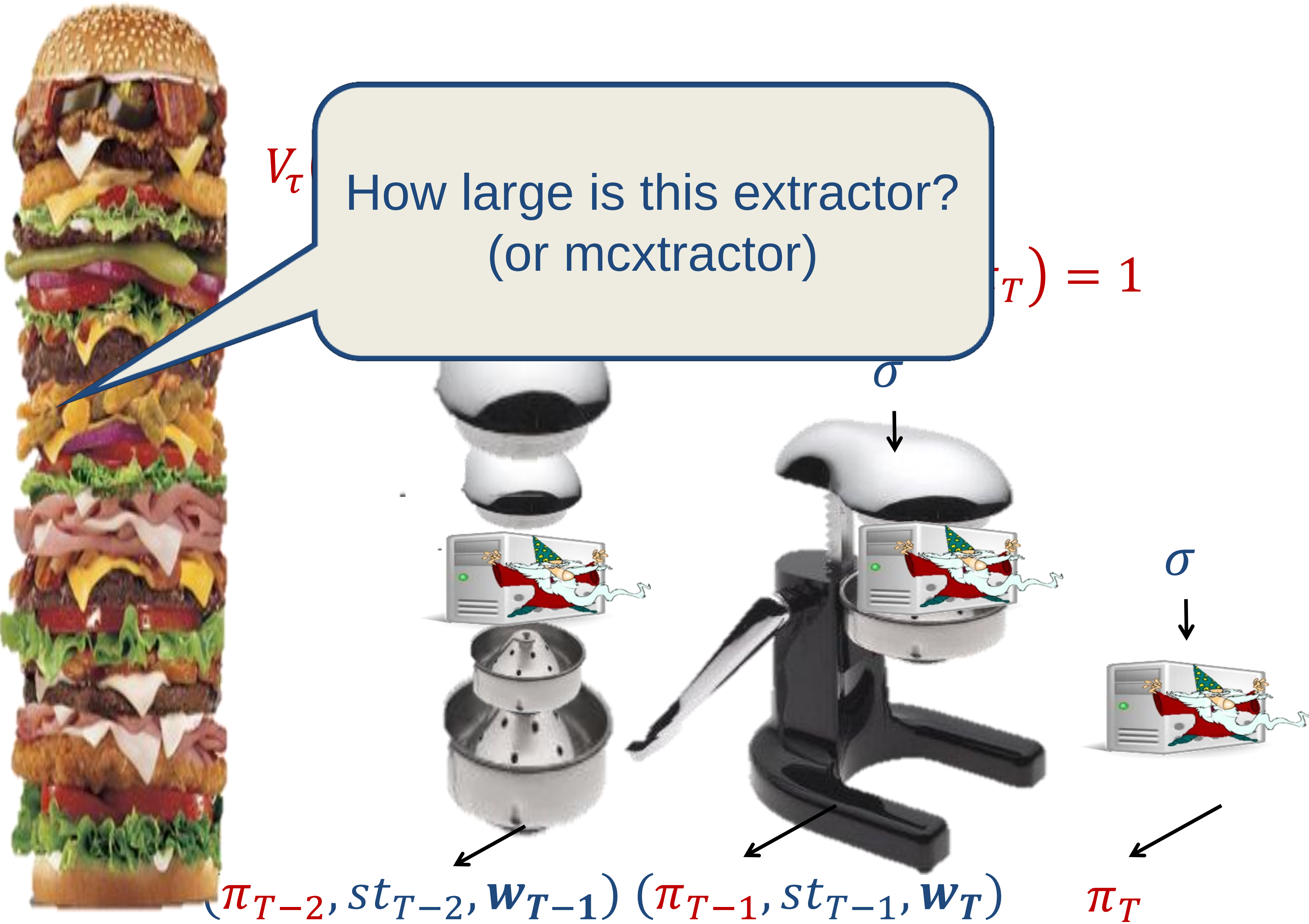
σ σ σ

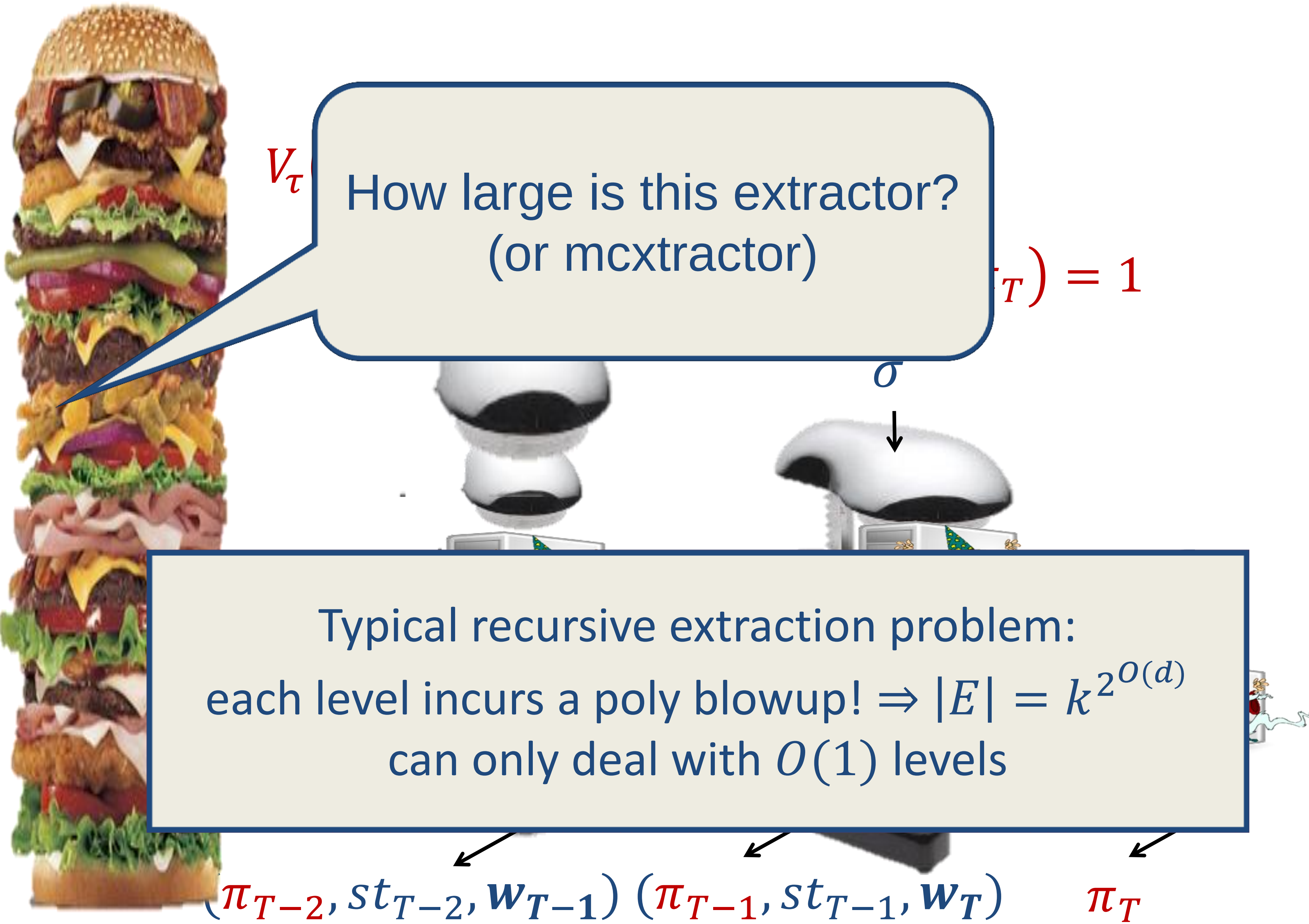
$(\pi_{T-2}, st_{T-2}, \boldsymbol{w}_{T-1})$ $(\pi_{T-1}, st_{T-1}, \boldsymbol{w}_T)$ π_T



$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-2}; \pi_{T-2}) = 1$
σ
$V_\tau(\boldsymbol{CH_{M_x}}; st_{T-1}; \pi_{T-1}) = 1$
σ
$V_\tau(\boldsymbol{CH_{M_x}}; st_T; \pi_T) = 1$
σ
σ
$(\pi_{T-2}, st_{T-2}, \boldsymbol{w_{T-1}})$ $(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T})$
π_T


$V_\tau(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_{T-1}; \pi_{T-1}) = 1$
σ
$V_\tau(\boldsymbol{CH}_{\boldsymbol{M_x}}; st_T; \pi_T) = 1$
σ
σ
$(\pi_{T-2}, st_{T-2}, \boldsymbol{w}_{\boldsymbol{T-1}})$
$(\pi_{T-1}, st_{T-1}, \boldsymbol{w}_{\boldsymbol{T}})$
π_T


How large is this extractor?
(or mcxtractor)
V_τ
$) = 1$
σ
σ
$(\pi_{T-2}, st_{T-2}, \boldsymbol{w_{T-1}})$ $(\pi_{T-1}, st_{T-1}, \boldsymbol{w_T})$ π_T

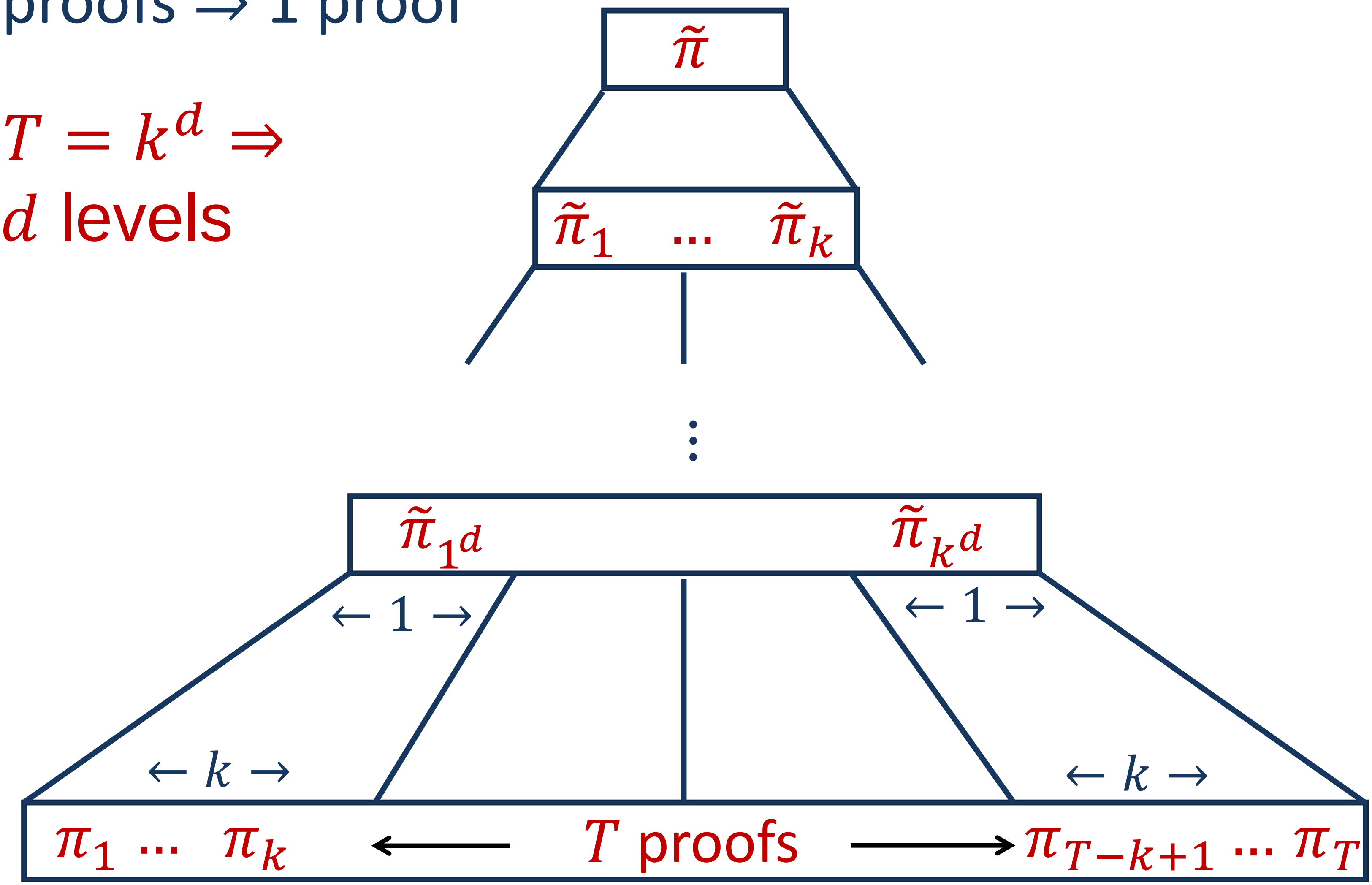


How large is this extractor?
(or mcxtractor)
$= 1$
σ
Typical recursive extraction problem:
each level incurs a poly blowup! $\Rightarrow |E| = k^{2^{O(d)}}$
can only deal with $O(1)$ levels
$(\pi_{T-2}, st_{T-2}, \mathbf{w}_{T-1})$ $(\pi_{T-1}, st_{T-1}, \mathbf{w}_T)$ π_T

A solution: aggregate proofs in a wide tree

k proofs $\Rightarrow$ 1 proof

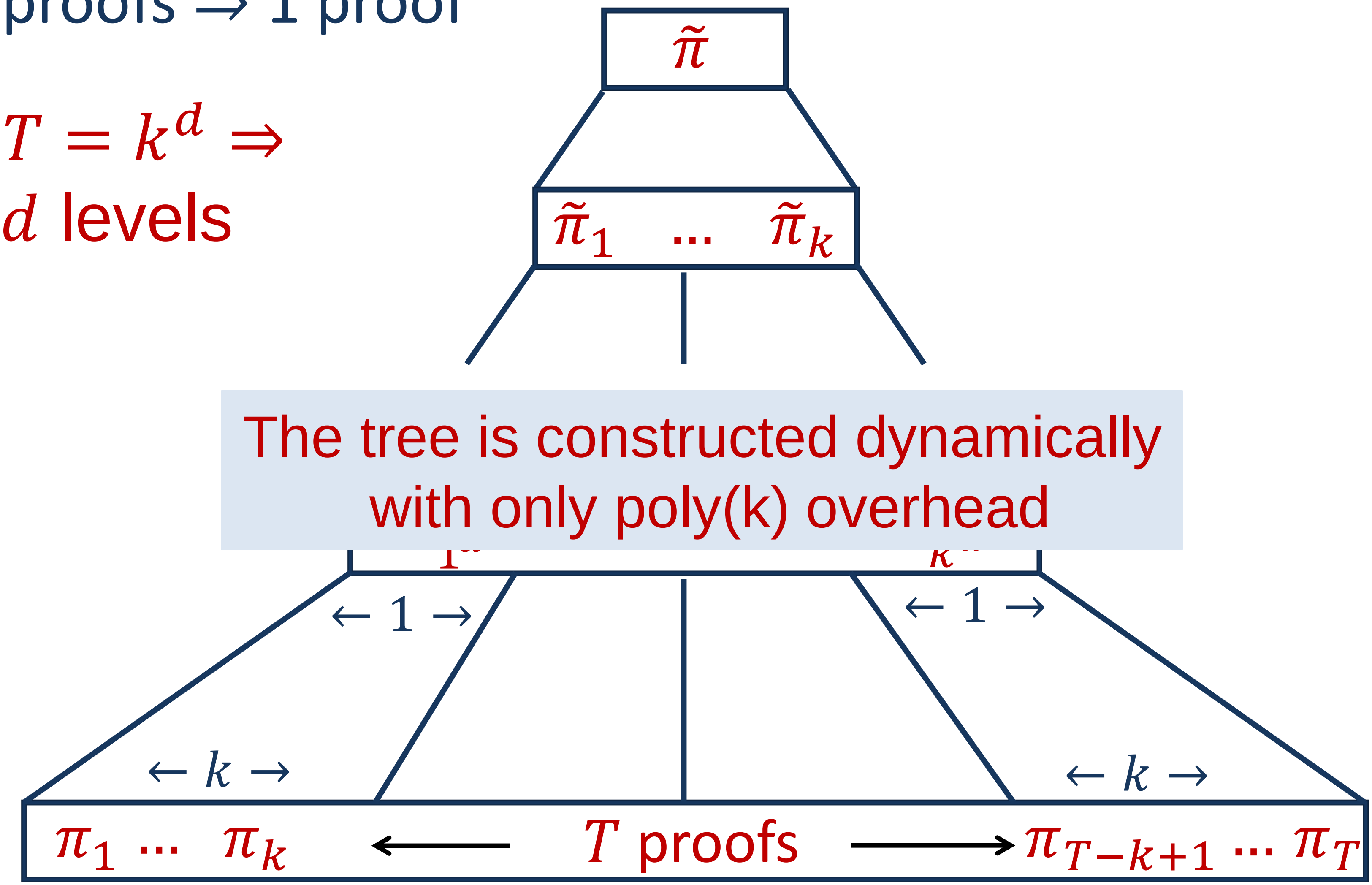
If $T = k^d \Rightarrow$
d levels

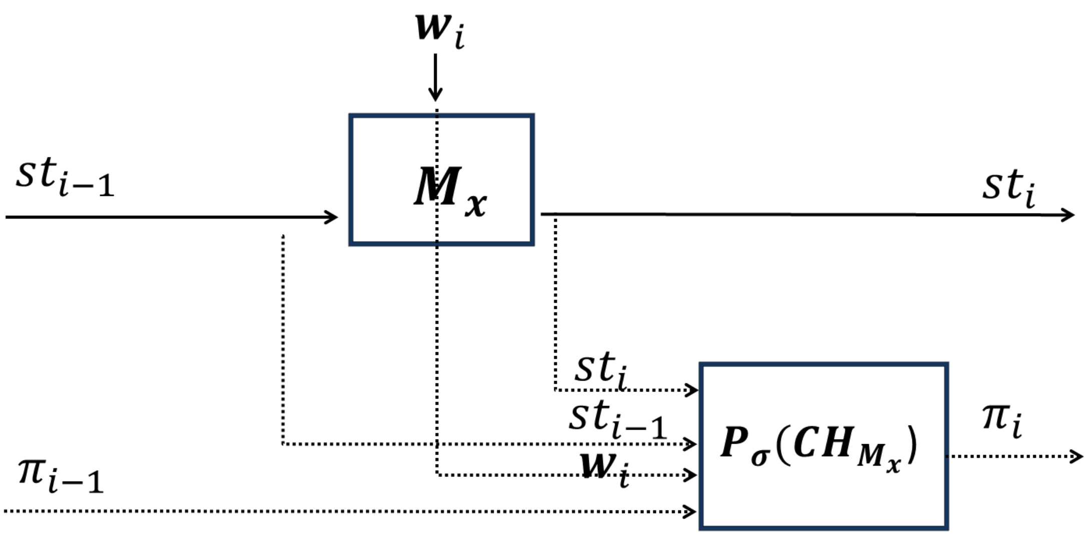


A solution: aggregate proofs in a wide tree

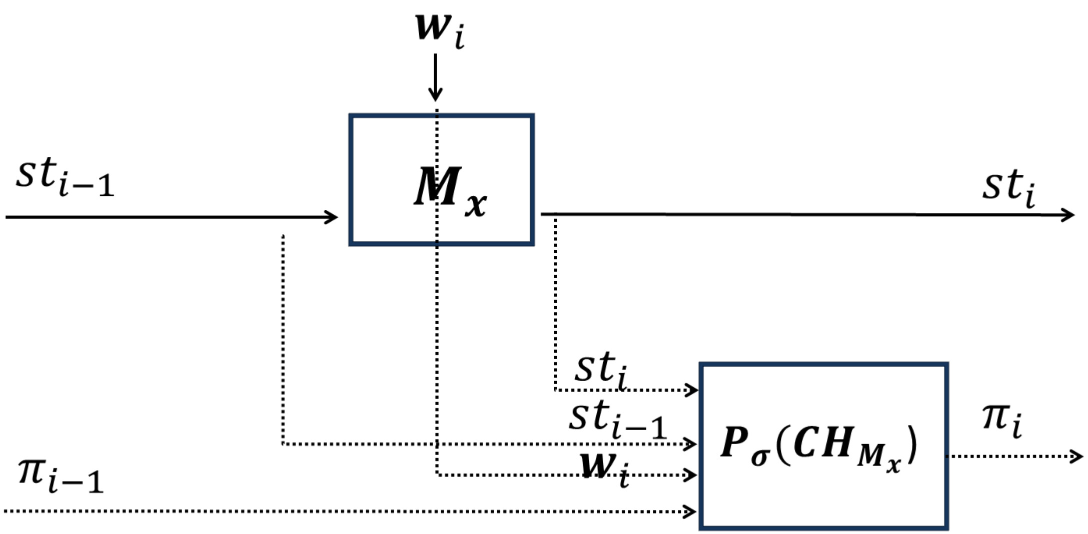
k proofs ⇒ 1 proof

If $T = k^d$ ⇒ d levels

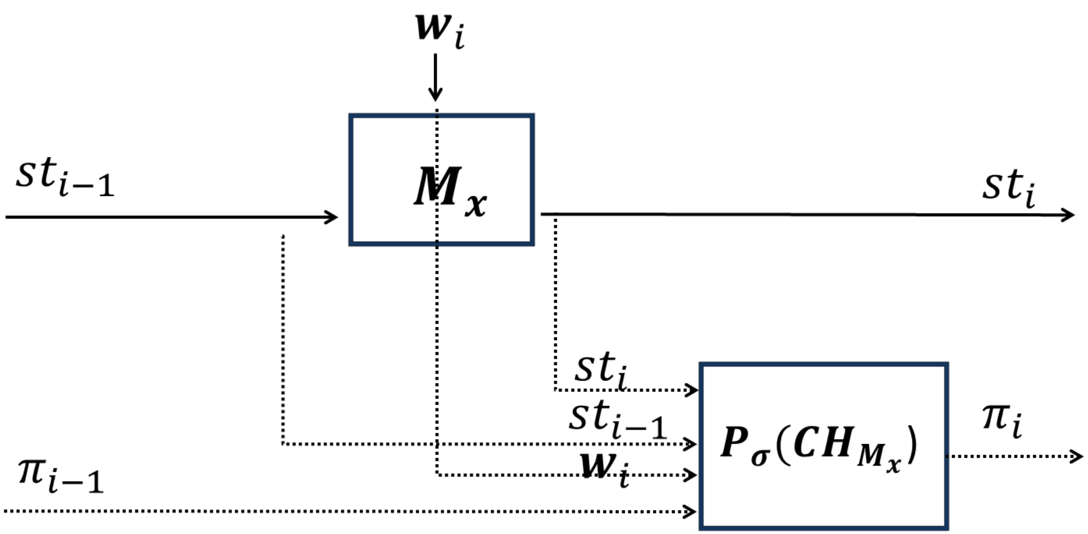




Is the resulting proof sound?

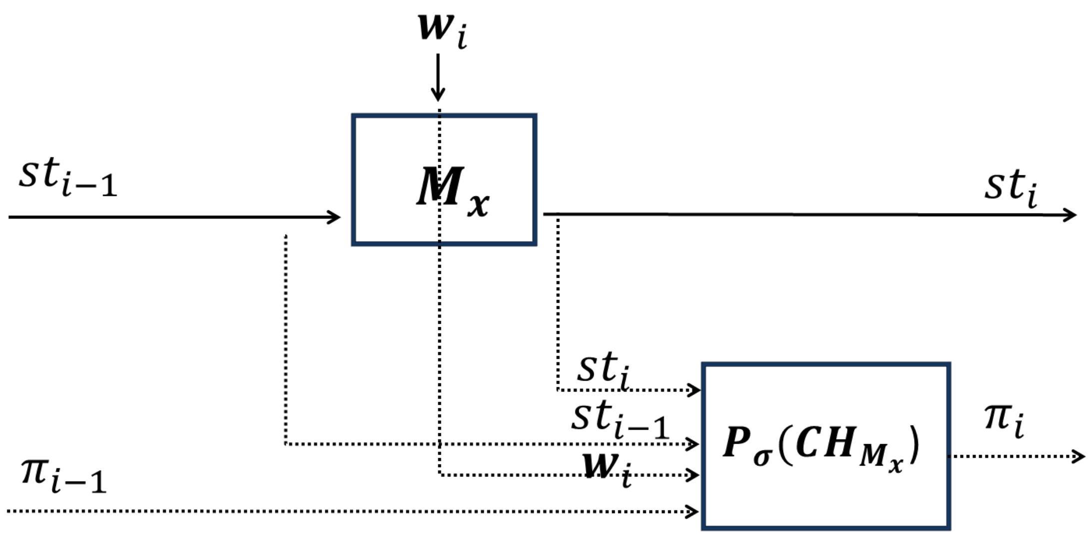


So Far: Preprocessing cost is proportional to single-step computation.



So Far: Preprocessing cost is proportional to single-step computation.

But how large is a single-step computation?



So Far: Preprocessing cost is proportional to single-step computation.

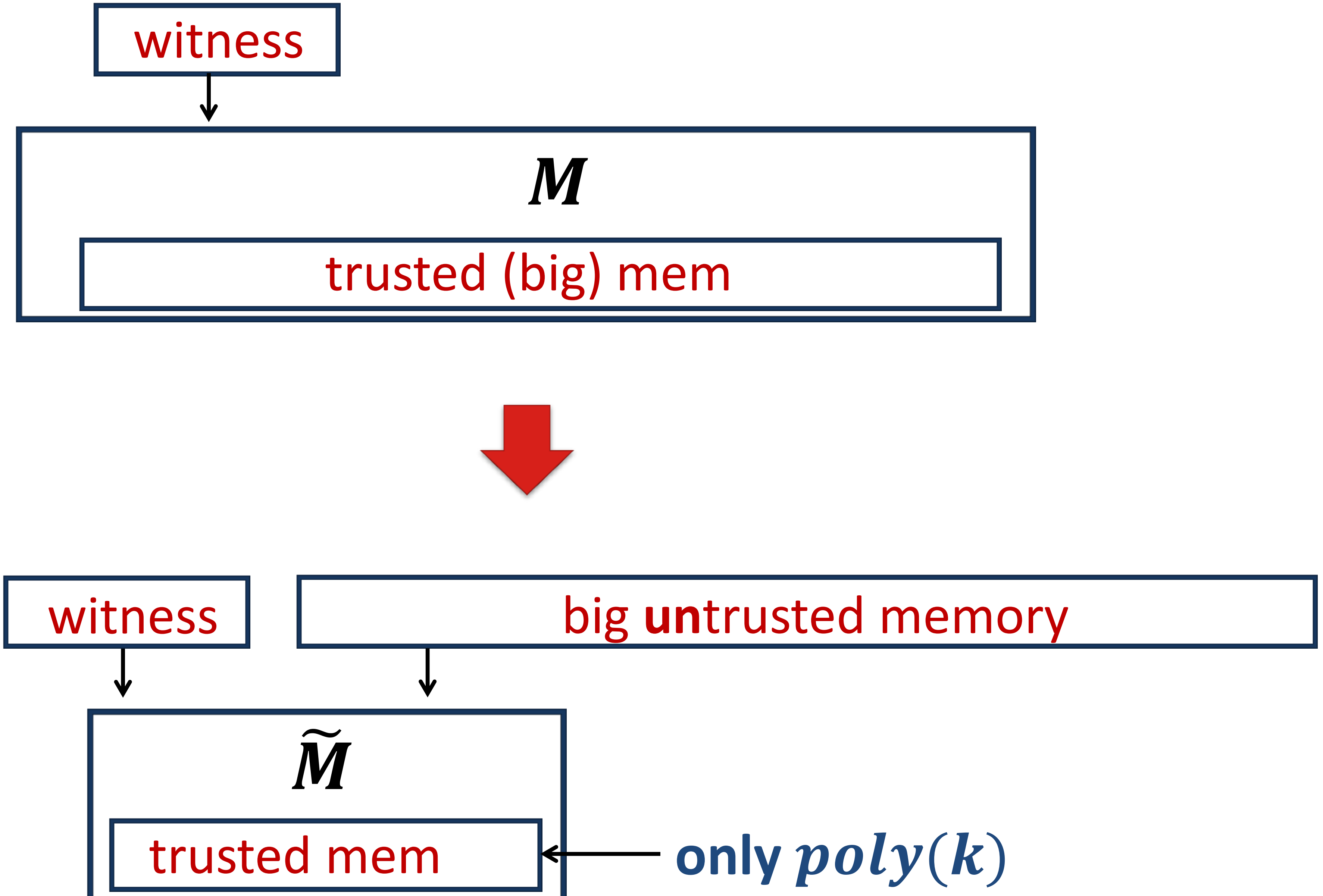
But how large is a single-step computation?

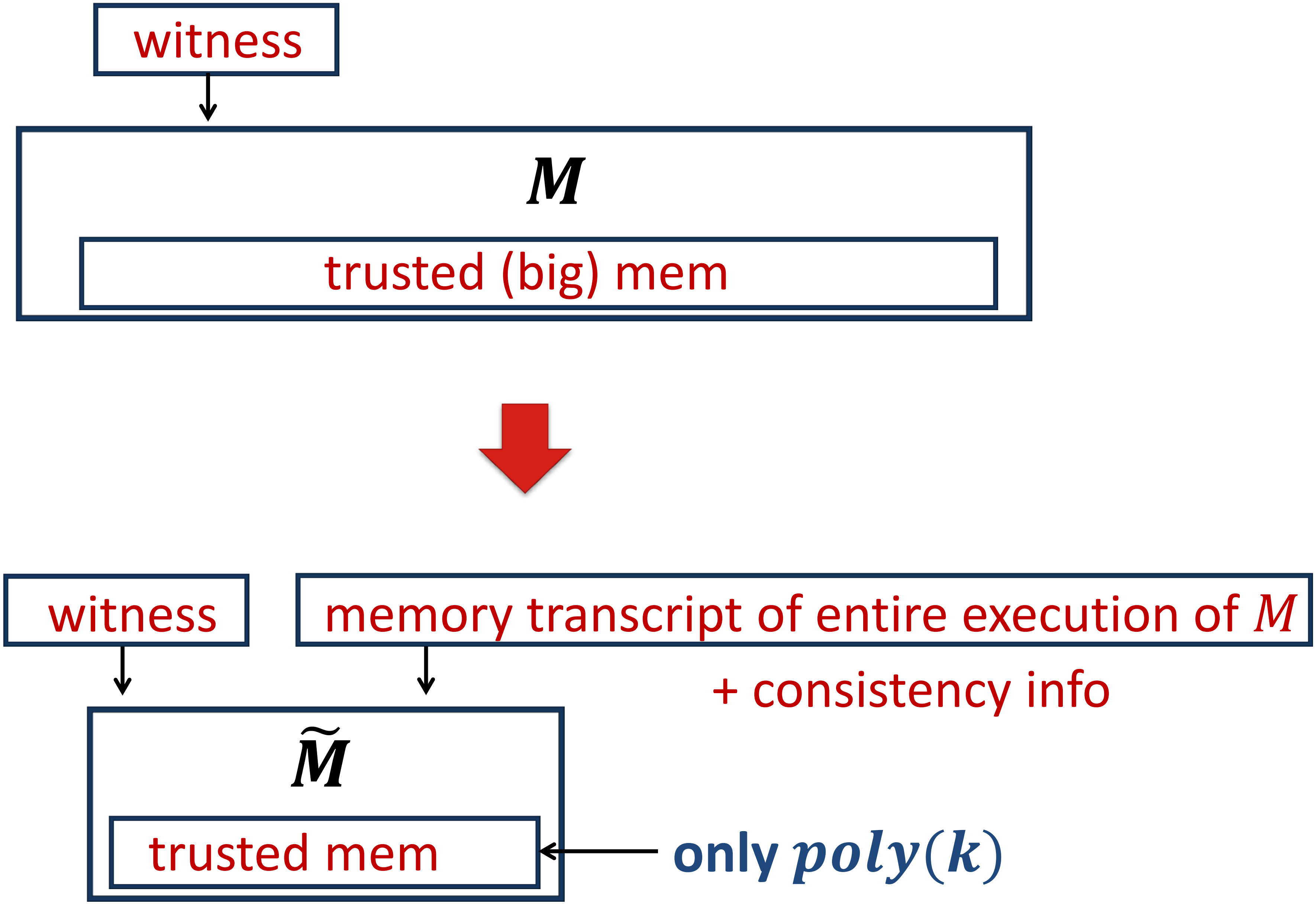
➔Bounded only by S, which can be as large as T…

➔Preprocessing stage can still be poly(T)…

Idea:
Move from machines with large memory to machines with:

- “small” trusted memory
- “big” untrusted memory


witness
M
trusted (big) mem
witness
big **un**trusted memory
$\widetilde{M}$
trusted mem
only $poly(k)$


witness
M
trusted (big) mem
witness
memory transcript of entire execution of M
+ consistency info
M̃
trusted mem
only poly(k)

A computational reduction using CRH

[Blum Evans Gemmell Kannan Naor 94, Ben-Sasson Chiesa Genkin Tromer 12]

$\boldsymbol{M} \xrightarrow{\;h\;} \widetilde{\boldsymbol{M}}_h$

$\boldsymbol{M}(\boldsymbol{x}, \boldsymbol{w})$ accepts **Or** h-collisions can be (eff.) extracted from mem $\Longleftarrow$ $\widetilde{\boldsymbol{M}}_h(\boldsymbol{x}, \boldsymbol{w}, mem)$ accepts

A computational reduction using CRH

[Blum Evans Gemmell Kannan Naor 94, Ben-Sasson Chiesa Genkin Tromer 12]

h

$\boldsymbol{M}$ → $\widetilde{\boldsymbol{M}}_h$

$\boldsymbol{M}(\boldsymbol{x}, \boldsymbol{w})$ accepts ← $\widetilde{\boldsymbol{M}}_h(\boldsymbol{x}, \boldsymbol{w}, mem)$ accepts

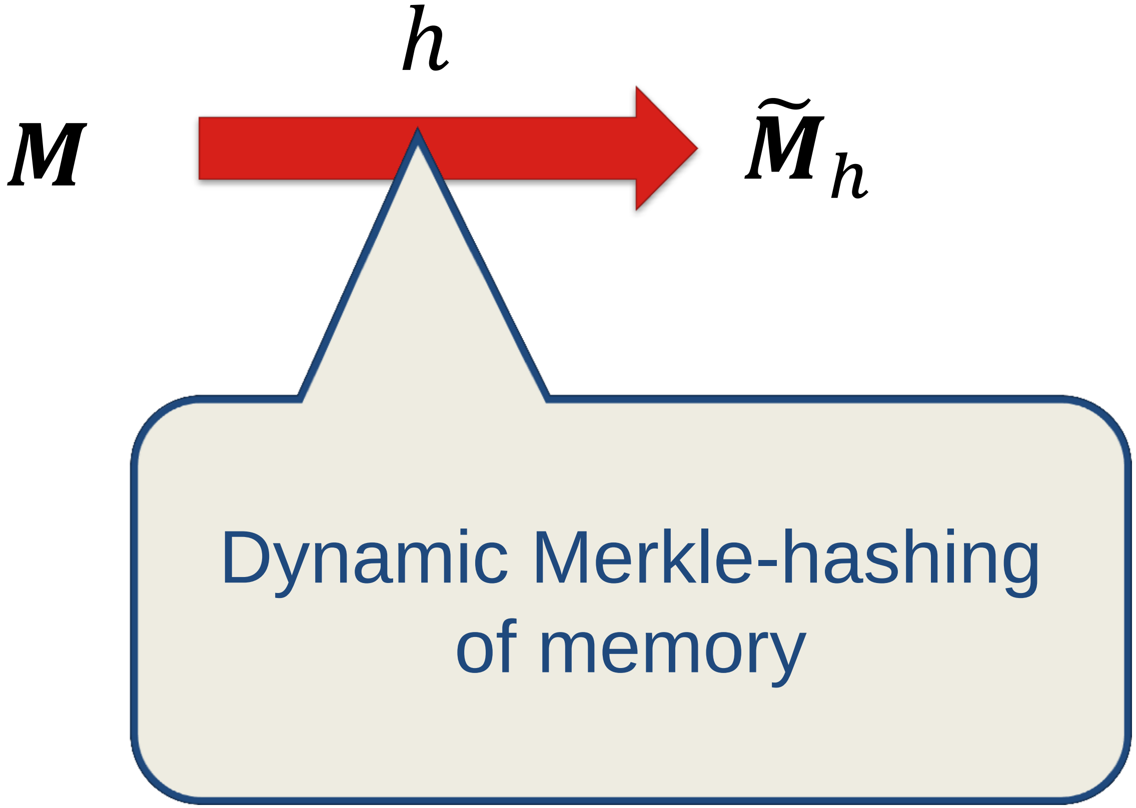
Or

h-collisions can be (eff.) extracted from mem

→ **$\exists mem$ not enough need knowledge**

A computational reduction using CRH

[Blum Evans Gemmell Kannan Naor 94, Ben-Sasson Chiesa Genkin Tromer 12]



$\widetilde{M}_h$ runs in time $T_M \cdot \text{poly}(k)$, space $\text{poly}(k)$ and *mem* computed from $(\boldsymbol{x}, \boldsymbol{w})$ in time $T_M \cdot \text{poly}(k)$ & space $S_M \cdot \text{poly}(k)$

➔ A single-step computation is now of size $\boldsymbol{poly_h(k)}$

(subsequent steps can be computed dynamically preserving time and space of original computation)

what's left?
…SNARK verification

Input: st_i witness: $(\pi_{i-1}, st_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\widetilde{\boldsymbol{M}}_{\boldsymbol{x}}$ accept.

else check:

$\widetilde{\boldsymbol{M}}_{\boldsymbol{x}}(st_{i-1}; \boldsymbol{w_i}) = st_i$

V_τ accepts π_{i-1} for statement $\boldsymbol{CH}_{\widetilde{\boldsymbol{M}}_{\boldsymbol{x}}}, st_{i-1}$

only $\mathrm{poly}_V(k)$,

independently of preprocessing limit

what's left?

...SNARK verification

Input: st_i witness: $(\pi_{i-1}, st_{i-1}, \boldsymbol{w_i})$

If st_i is initial state of $\widetilde{\boldsymbol{M}}_x$ accept.

else check:

$\widetilde{\boldsymbol{M}}_x(st_{i-1}; \boldsymbol{w_i}) = st_i$

V_τ accepts π_{i-1} for statement $\boldsymbol{CH}_{\widetilde{\boldsymbol{M}}_x}, st_{i-1}$

only $\text{poly}_V(k)$,

independently of preprocessing limit

⇒ budget only for $\boldsymbol{poly_V(k)}$+$\boldsymbol{poly_h(k)}$

Bye Bye
Long Preprocessing…

Part I:

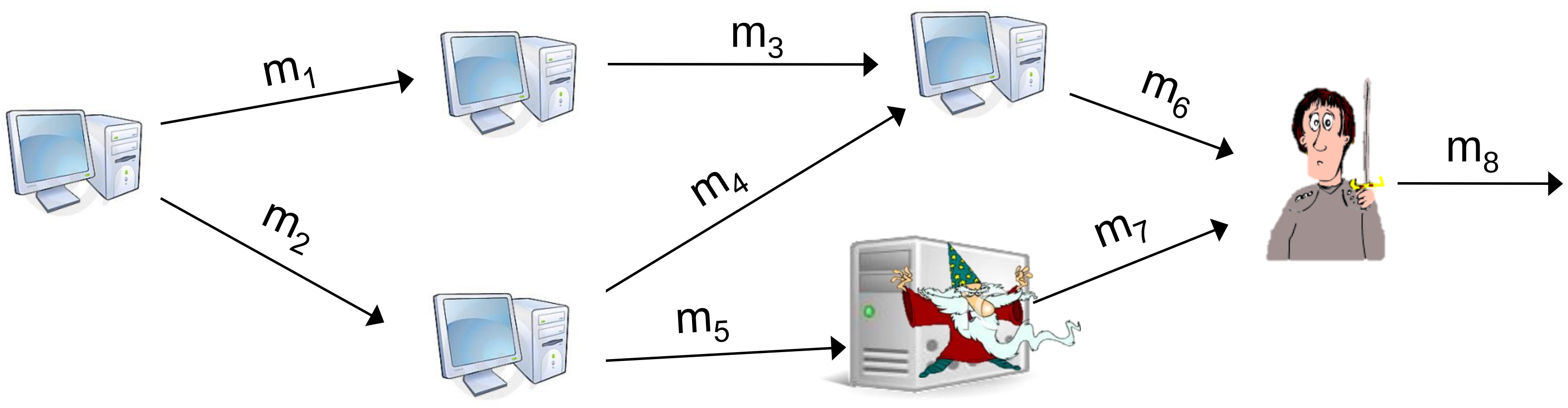
How to Bootstrap a SNARK
in Public

Part II:

Part I (again) and Beyond
with Proof Carrying Data

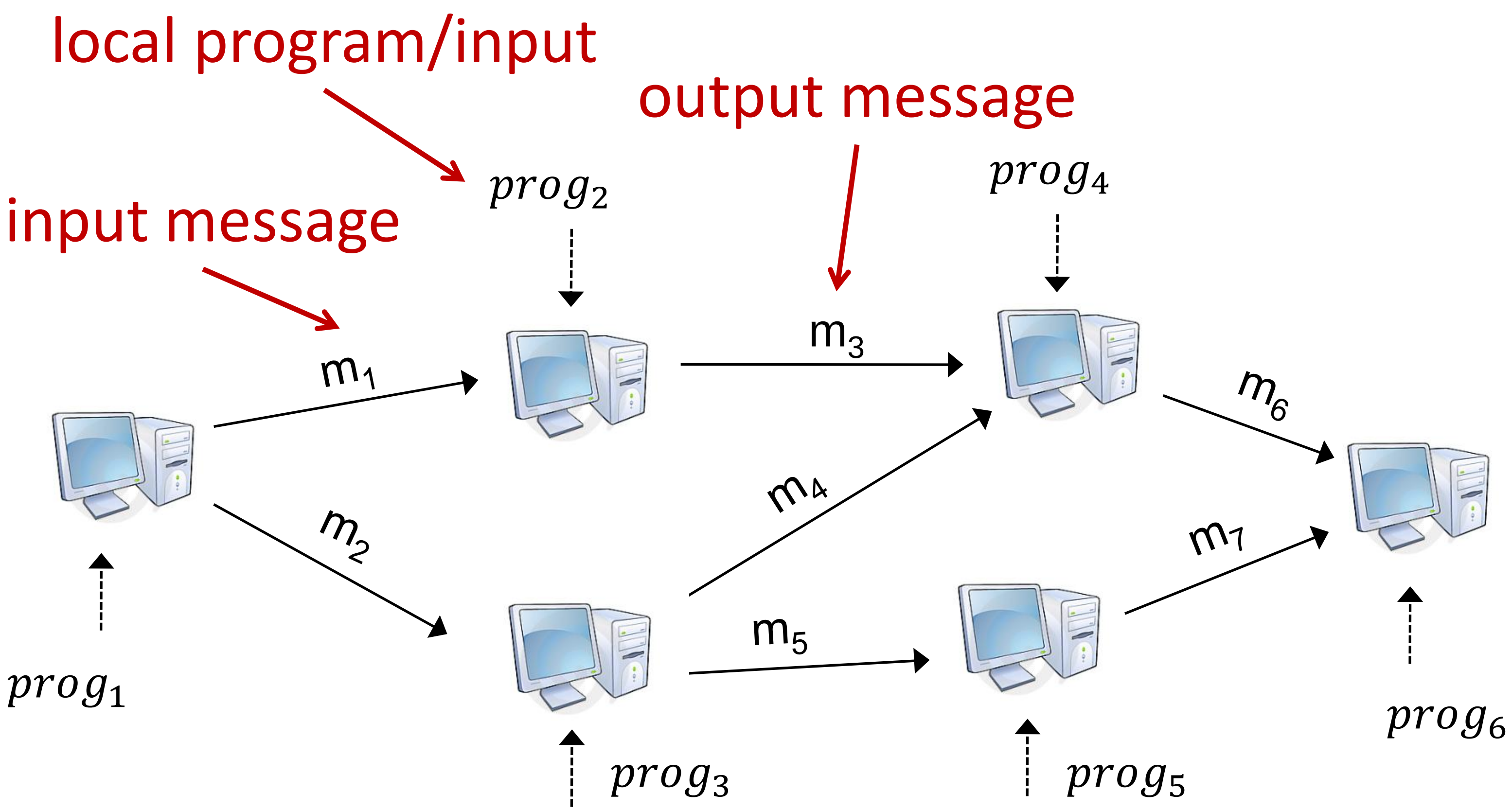
In SNARKs: one prover and one verifier

But sometimes in life…



Computations involve many parties
each party has its own:
role, capabilities, friends, enemies,…

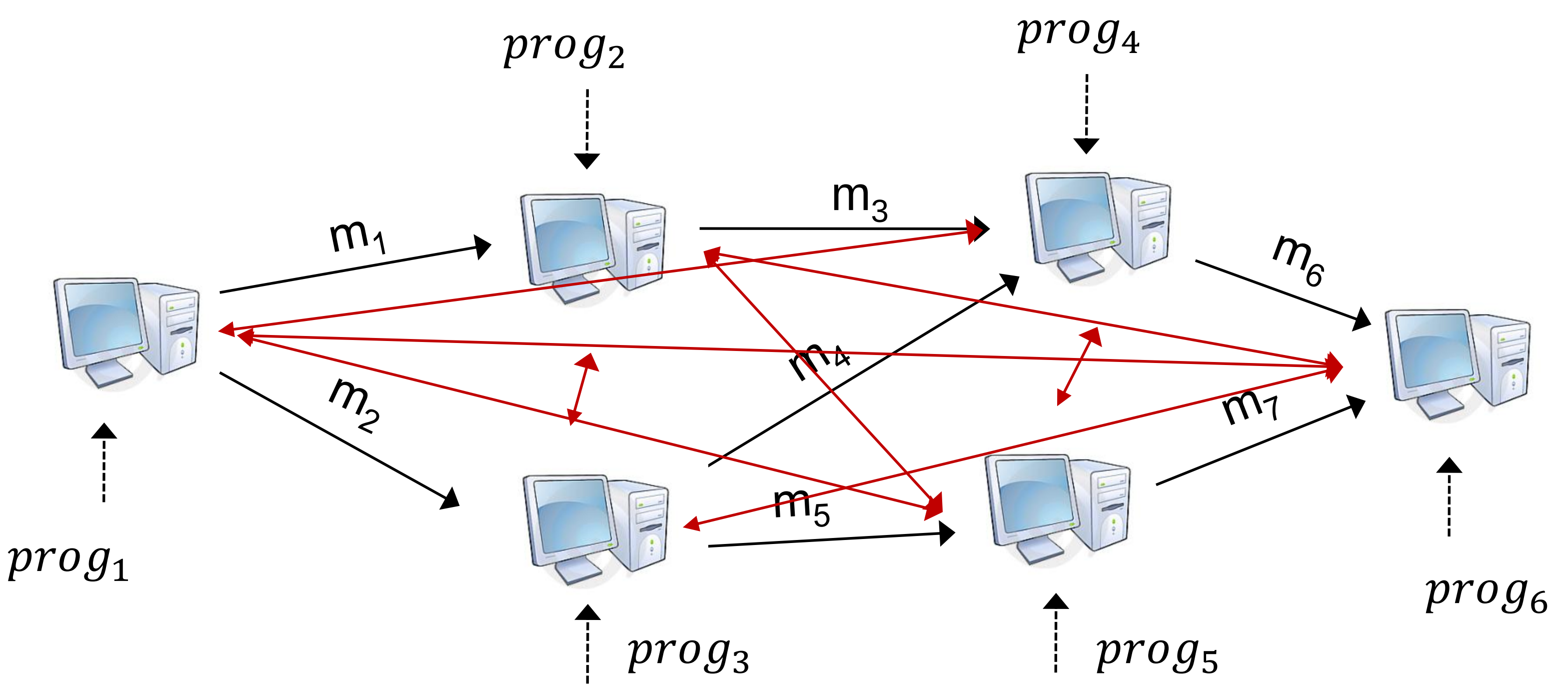
How can we enforce general correctness properties of distributed computations?



Use MPC?

enforce **any** property of **all** the inputs/outputs of all parties

but: large overhead: all parties must communicate with each other (necessary, e.g. Byzantine agreement)

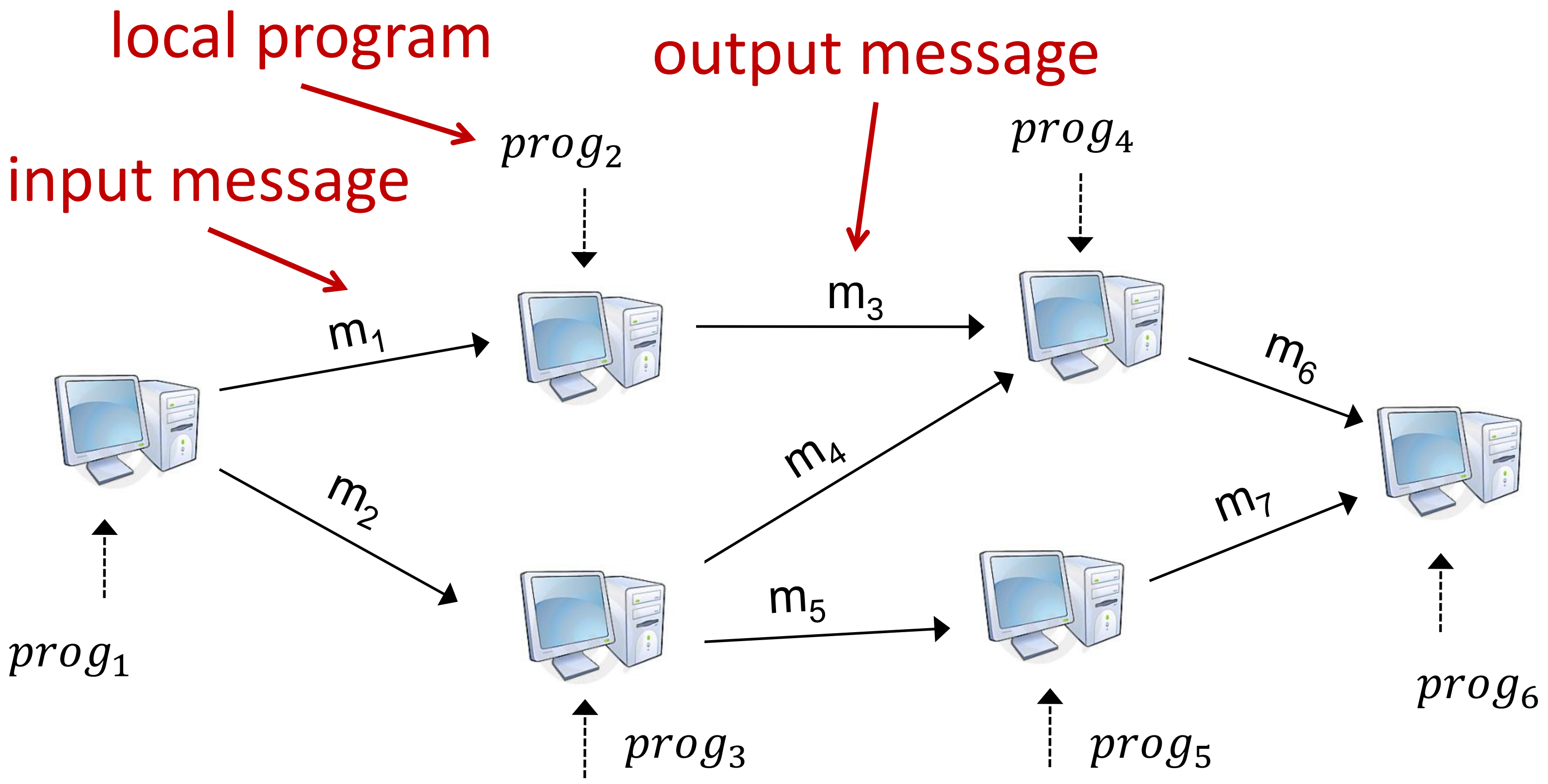


A relaxed question:
how to enforce **local** properties?

Local property = property of the view of a single node

Example: ensure that the program executed at every node was signed by system admin

if property holds **everywhere** → global meaning



Proof Carrying Data (PCD)
[Chiesa Tromer 10]

<u>Goal</u>:

Guarantee “local properties” **while respecting the original computation**:

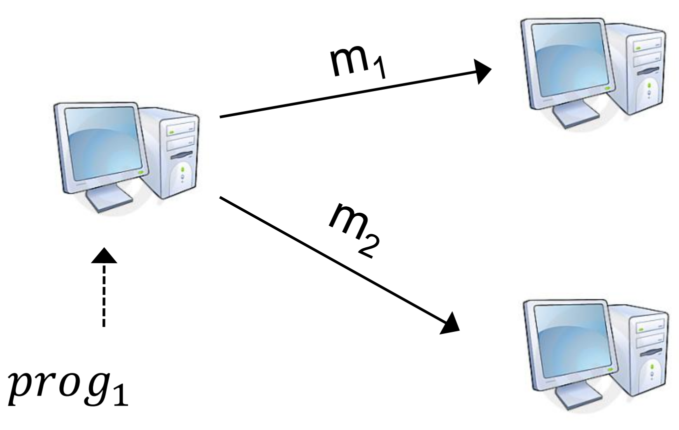
- preserve communication graph
- minimal computational overhead

The original computation

- Can be viewed as a DAG evolving over time
- nodes have input and output messages
  + a local program (with embedded inputs).

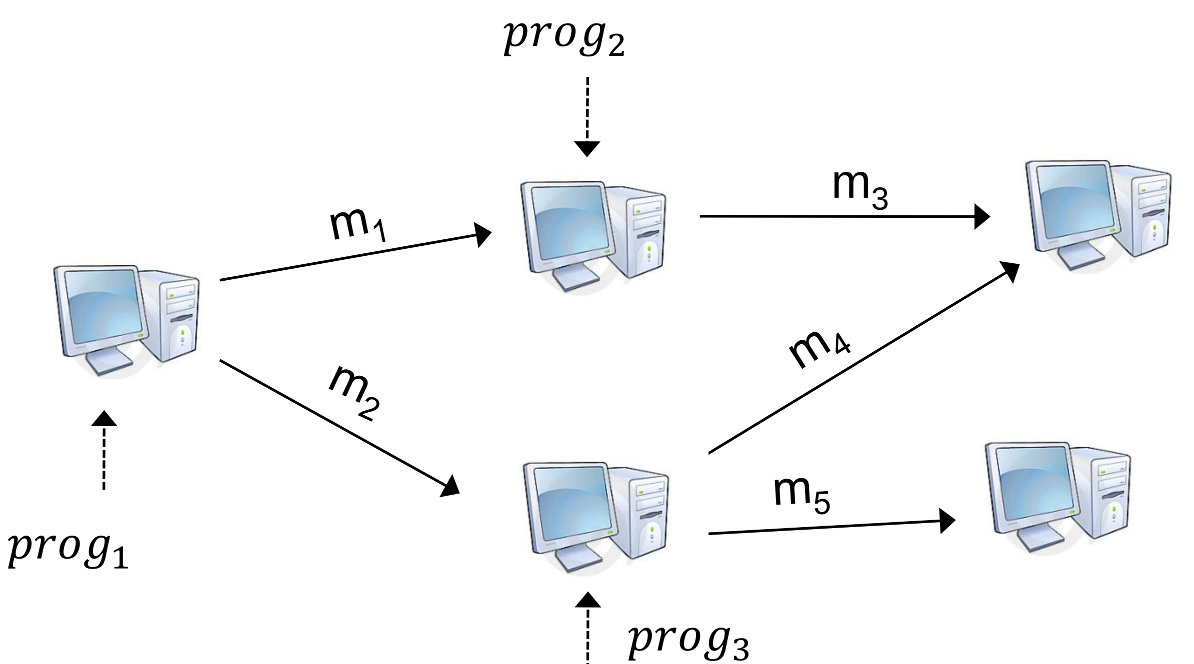
The original computation

- Can be viewed as a DAG evolving over time
- nodes have input and output messages
  + a local program (with embedded inputs).



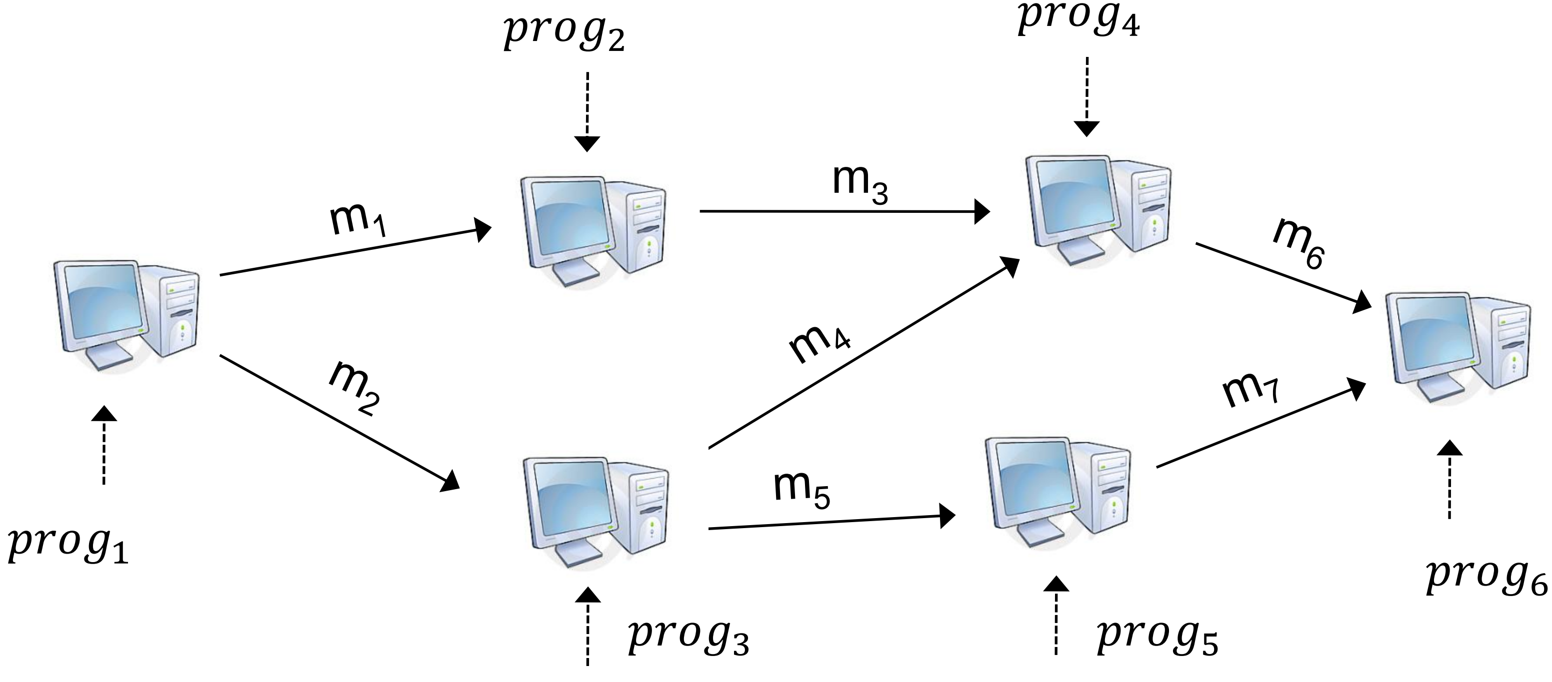
The original computation

- Can be viewed as a DAG evolving over time
- nodes have input and output messages + a local program (with embedded inputs).



The original computation

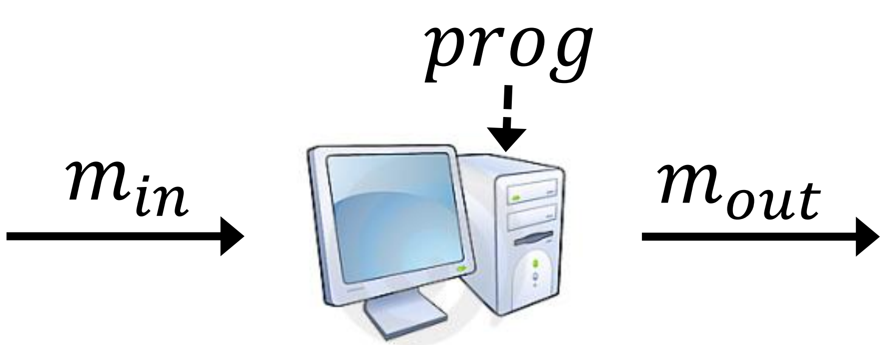
- Can be viewed as a DAG evolving over time
- nodes have input and output messages
  + a local program (with embedded inputs).



Local properties as $\boldsymbol{C}$-compliance

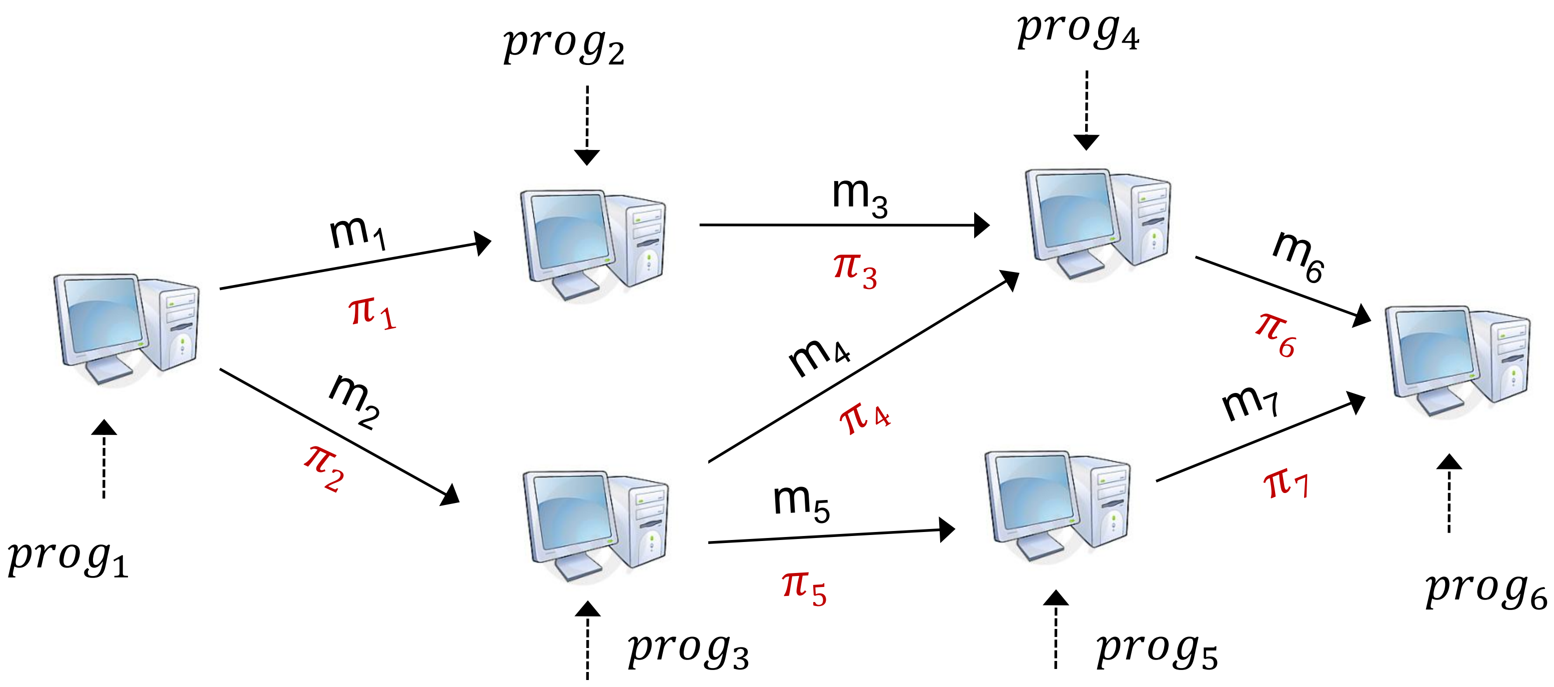
$\boldsymbol{C}(prog, m_{in}, m_{out})$ is a predicate specifying a local property, e.g.:

- $\boldsymbol{C}_{adm}$: "$prog = (M, s)$ where s is an admin signature on M and $M(m_{in}) = m_{out}$"
- $\boldsymbol{C}_{\boldsymbol{JVM}}$: "$prog$ is a JAVA program and $JVM(prog, m_{in}) = m_{out}$"
- $\boldsymbol{C}_{M_x}$: "$prog = \boldsymbol{w_i}, m_{in} = st_{i-1}$, $m_{out} = st_i$ and $\boldsymbol{M_x}(st_{i-1}, \boldsymbol{w_i}) = st_i$ "



A PCD system

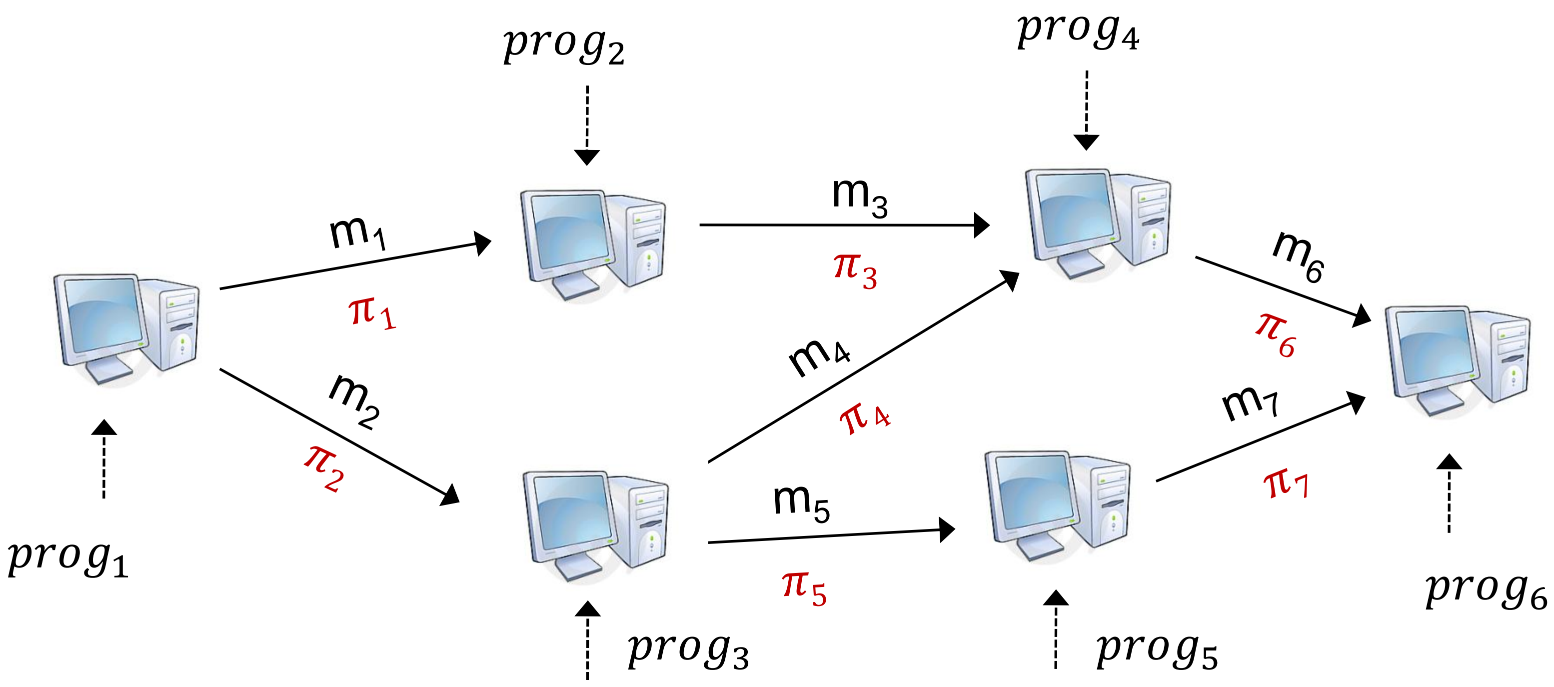
- compile on-the-fly original computation
- (short) proofs are appended to messages



A PCD system

- compile on-the-fly original computation
- (short) proofs are appended to messages

Note: not all properties can be verified this way.
Eg, verifying that $m_1 = m_2$ requires additional interaction.



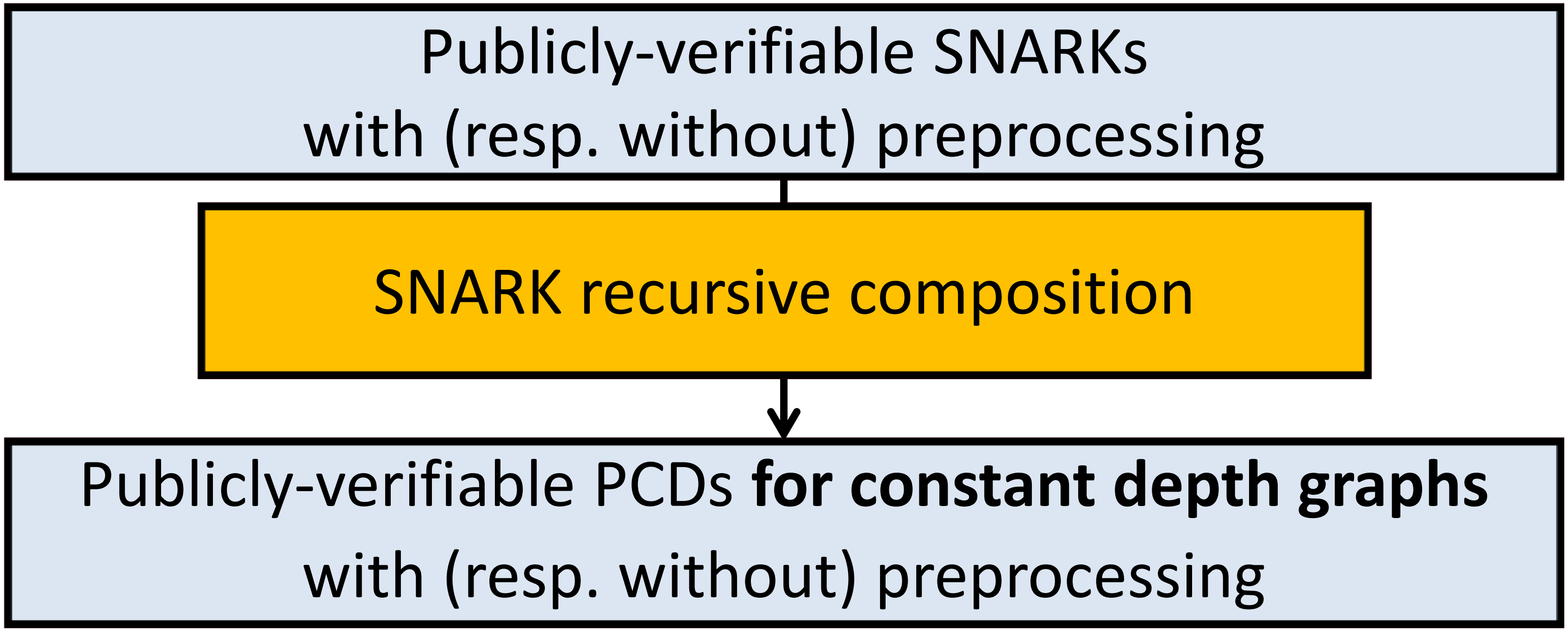
How to construct PCDs?

[CT10]: Using an abstract signature card

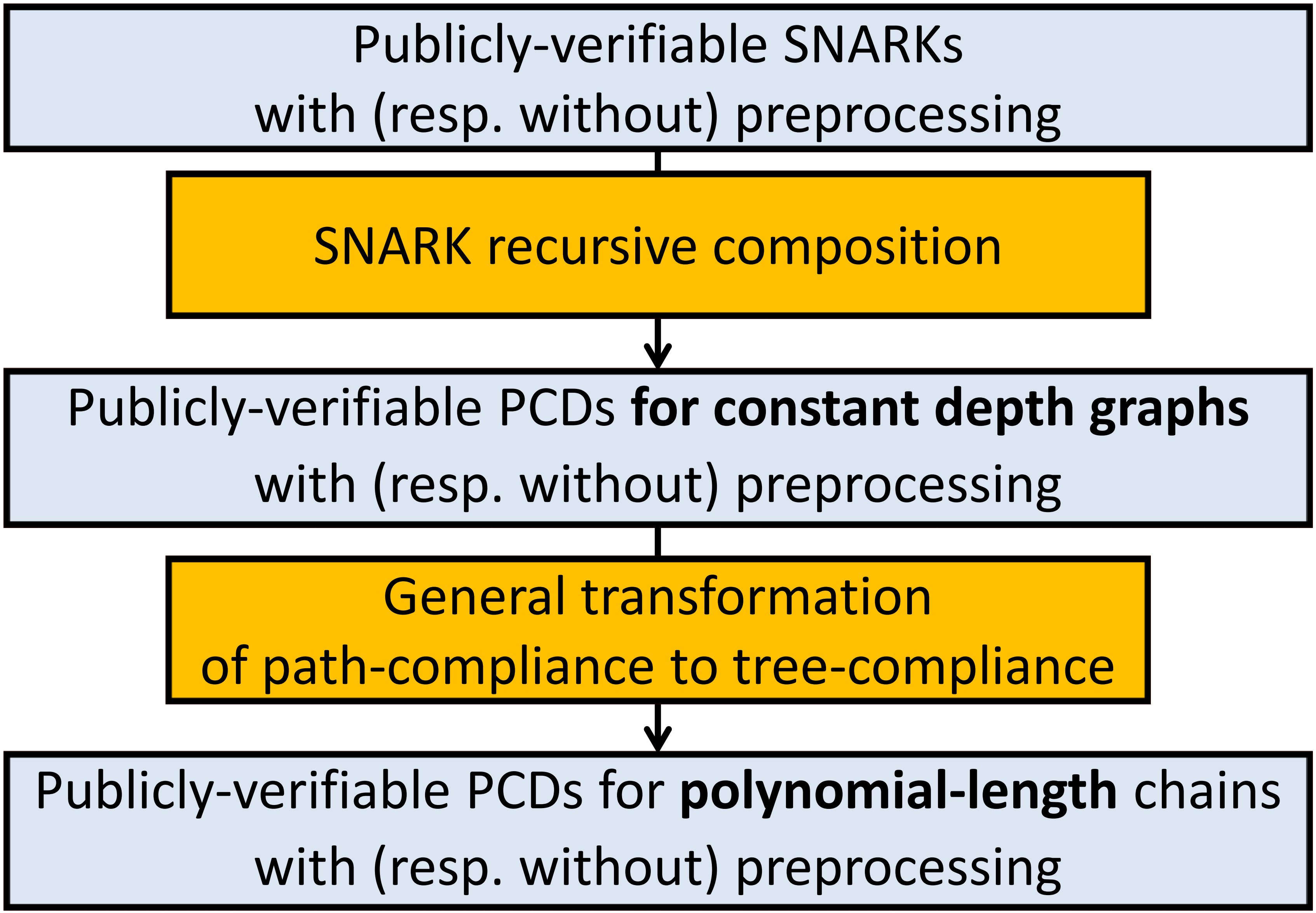
How to construct PCDs?

This work: SNARK composition

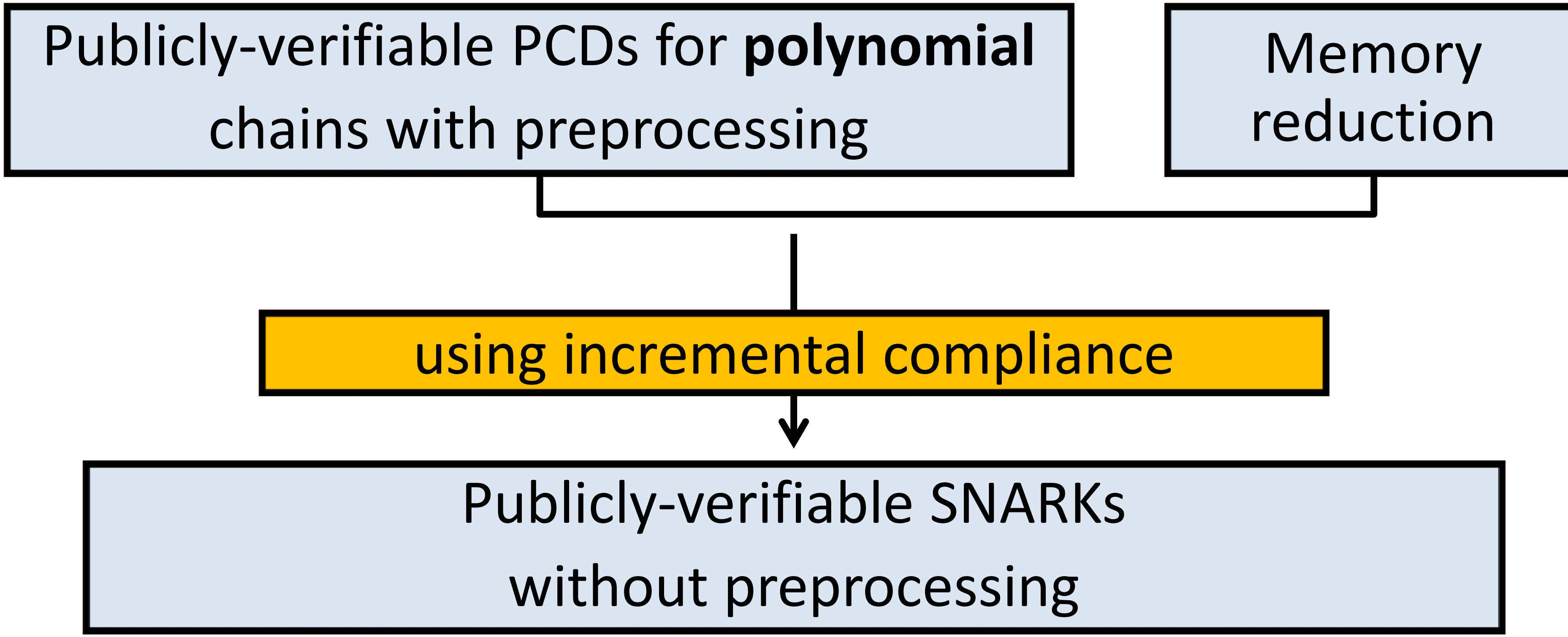
Results (revisited): General transformations



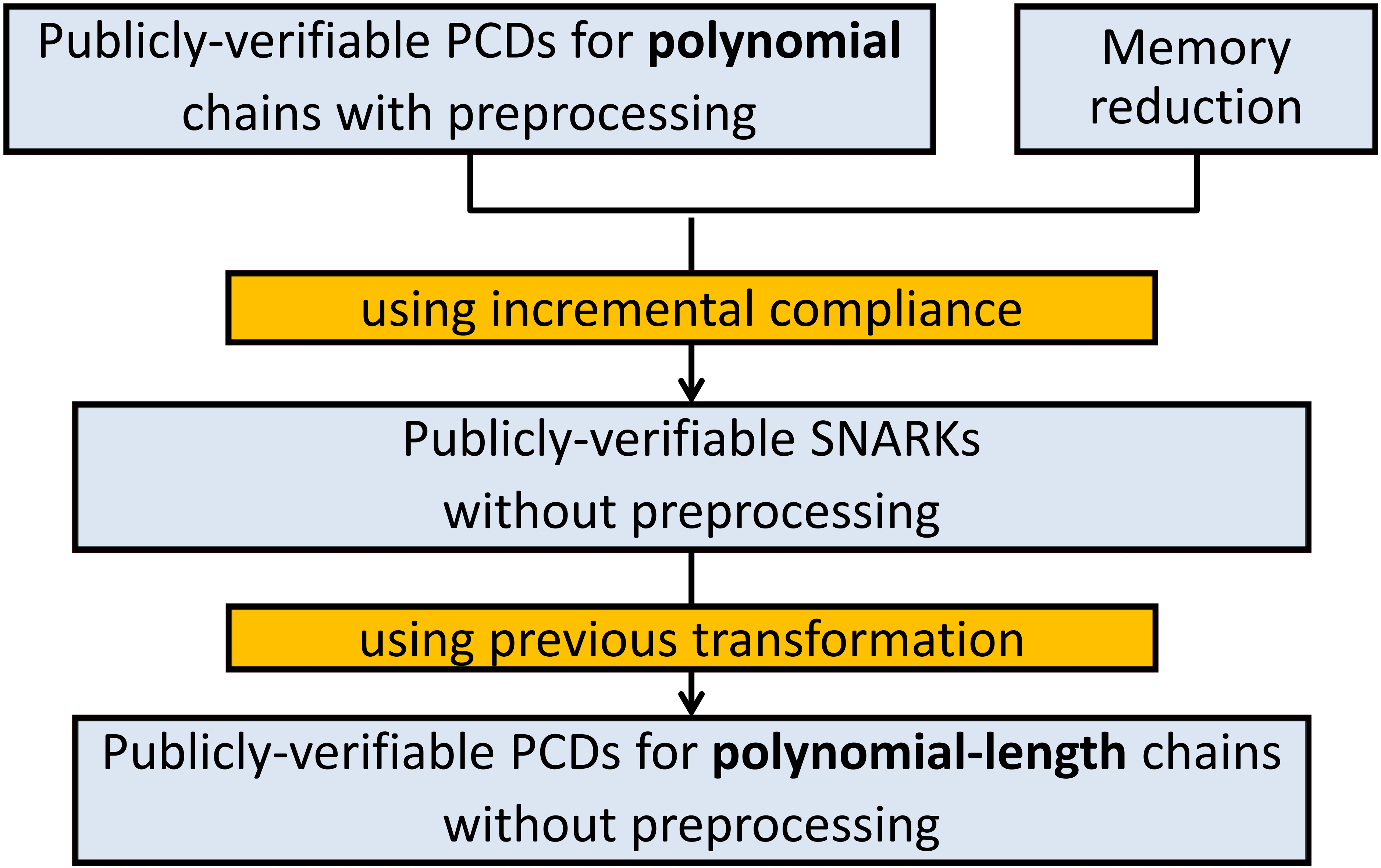
Results (revisited): General transformations



Results (revisited): Eliminating expensive preprocessing



Results (revisited): Eliminating expensive preprocessing



A bonus:
privately-verifiable SNARKs also compose!

A bonus:
privately-verifiable SNARKs also compose!

To compose SNARKs we used public-verifiability
proved "I verified a SNARK"

A bonus: privately-verifiable SNARKs also compose!

To compose SNARKs we used public-verifiability
proved "I verified a SNARK"

Looks surprising… but doable (using FHE).

➔ All the PCD results have their privately-verifiable analogs

Question:

which security goals we express using the PCD language?

We've seen some examples
others include: targeted-malleability [BSW11], computing on authenticated Data,…

Other properties?