FOUNDATIONAL RESPONSE-TIME ANALYSIS AS EXPLAINABLE EVIDENCE OF TIMELINESS

ECRTS 2022

8 June 2022

Marco Maida, **Sergey Bozhko**, and Björn B. Brandenburg

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

MAX-PLANCK-GESELLSCHAFT

THIS PAPER IN A NUTSHELL

We introduce

Foundational Response-Time Analysis,

a new way of implementing response-time analysis tools

THIS PAPER IN A NUTSHELL

We introduce

Foundational Response-Time Analysis,

a new way of implementing response-time analysis tools

that: 1. is inherently **trustworthy** – by design free of analysis or implementation bugs affecting the safety of the results,

THIS PAPER IN A NUTSHELL

We introduce

Foundational Response-Time Analysis,

a new way of implementing response-time analysis tools

that:

1. is inherently **trustworthy** – by design free of analysis or implementation bugs affecting the safety of the results,
2. does **not** require verifying the analysis tool itself, *and*

THIS PAPER IN A NUTSHELL

We introduce

Foundational Response-Time Analysis,

a new way of implementing response-time analysis tools

that:

1. is inherently **trustworthy** – by design free of analysis or implementation bugs affecting the safety of the results,
2. does **not** require verifying the analysis tool itself, *and*
3. produces **explainable evidence** of timeliness, which can be assessed independently of the tool itself.

+ a **proof of concept** that scales to large workloads (parameter magnitudes & task counts)

…IN A NUTSHELL

We introduce

Foundational Response-Time Analysis,

a new way of implementing response-time analysis tools

that:

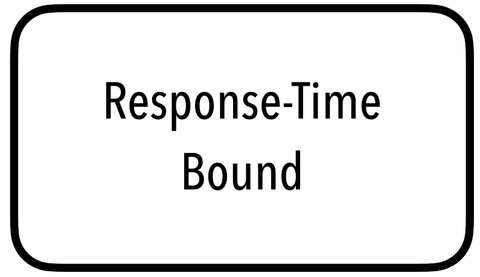
1. is inherently **trustworthy** – by design free of analysis or implementation bugs affecting the safety of the results,
2. does **not** require verifying the analysis tool itself, *and*
3. produces **explainable evidence** of timeliness, which can be assessed independently of the tool itself.

MOTIVATION

What is the issue with conventional response-time analysis?

RESPONSE-TIME ANALYSIS (RTA)

The goal of an **RTA** is to obtain **safe response-time bounds**



as ***evidence of temporal correctness,***
e.g., for use in the certification of safety-critical systems.

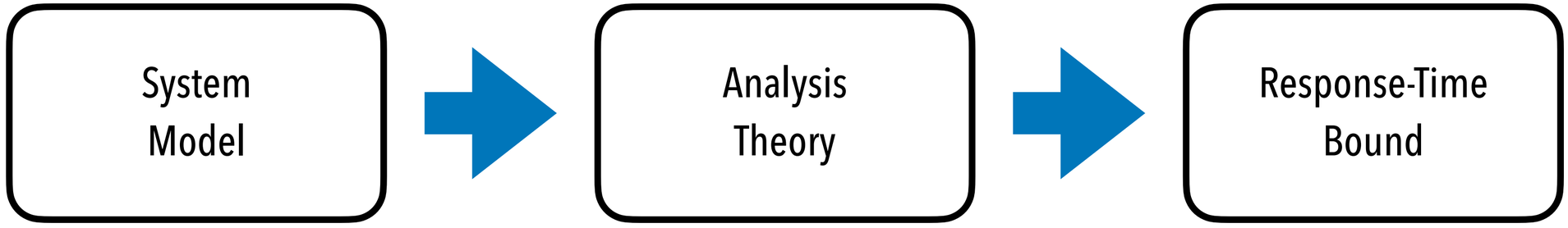
RESPONSE-TIME ANALYSIS (RTA)

The goal of an **RTA** is to obtain **safe response-time bounds**

as ***evidence of temporal correctness,***
e.g., for use in the certification of safety-critical systems.

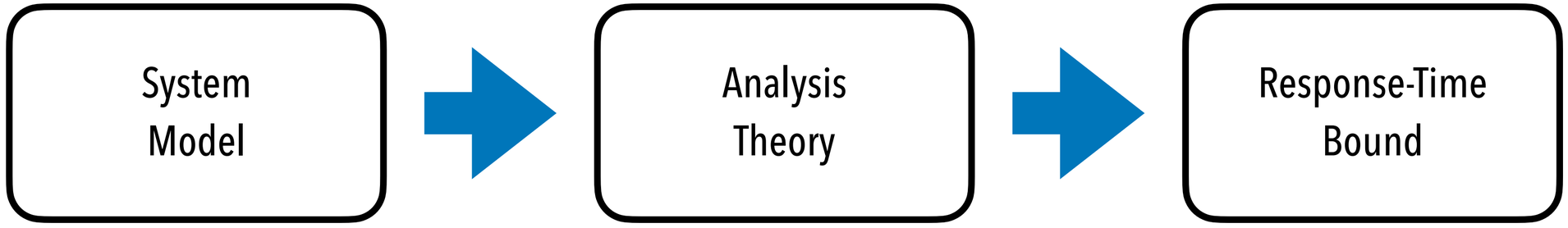
RESPONSE-TIME ANALYSIS (RTA)

The goal of an **RTA** is to obtain **safe response-time bounds**



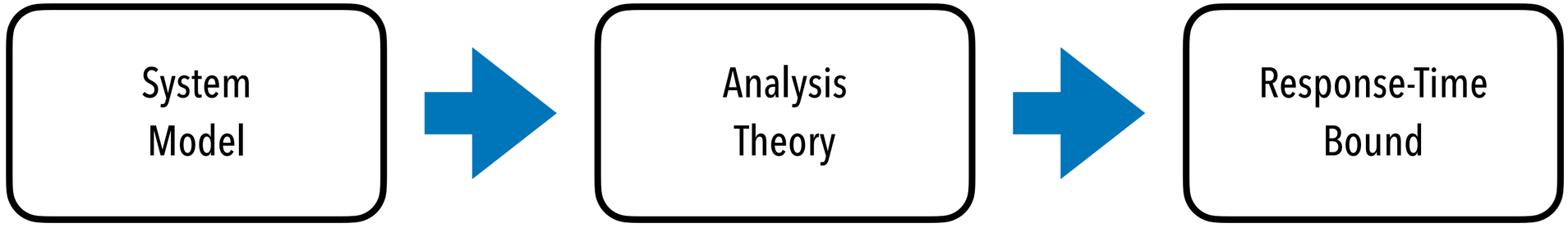
as ***evidence of temporal correctness,***
e.g., for use in the certification of safety-critical systems.

ISSUE 1: RTA IS NOT EXPLAINABLE



ISSUE 1: RTA IS NOT EXPLAINABLE

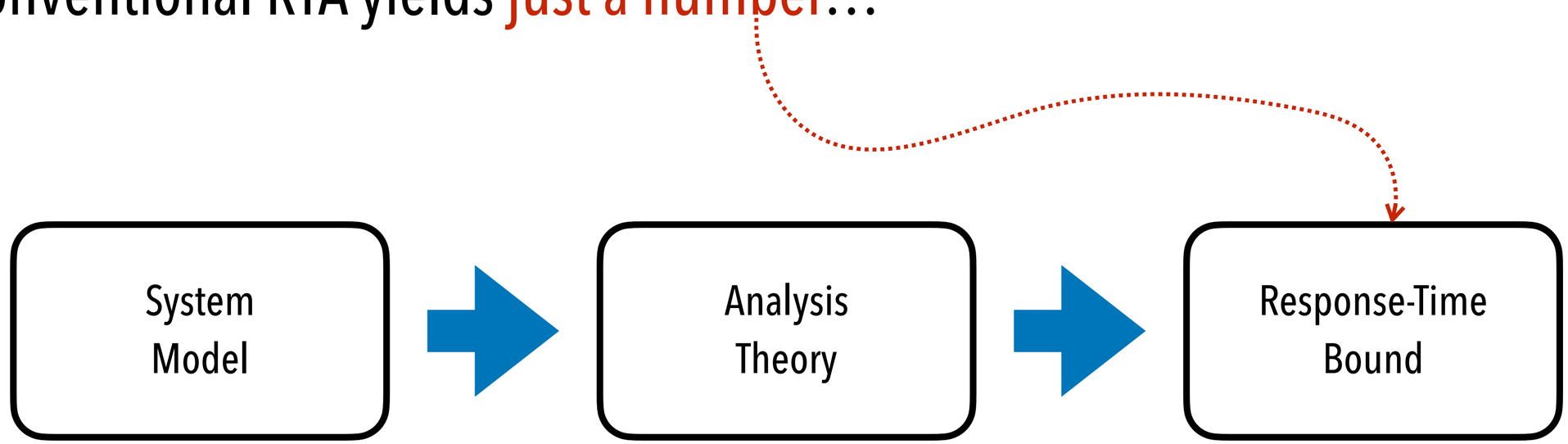
➔ *the ability for a human evaluator to **inspect** and **understand** the result of a computation*



ISSUE 1: RTA IS NOT EXPLAINABLE

➔ *the ability for a human evaluator to **inspect** and **understand** the result of a computation*

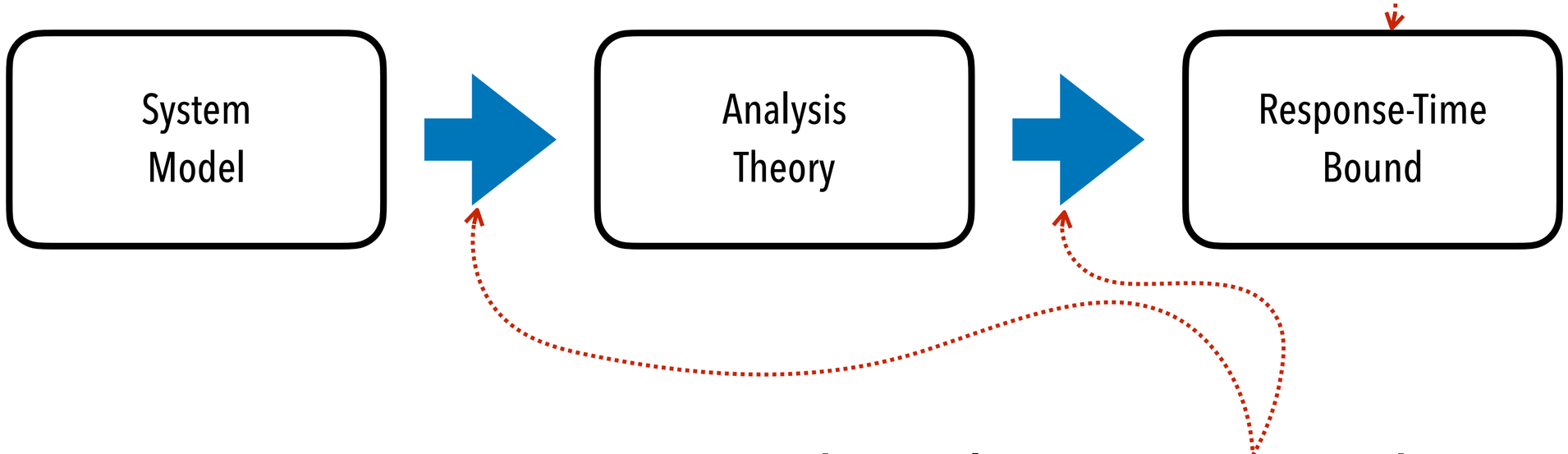
Conventional RTA yields just a number…



ISSUE 1: RTA IS NOT EXPLAINABLE

➔ *the ability for a human evaluator to **inspect** and **understand** the result of a computation*

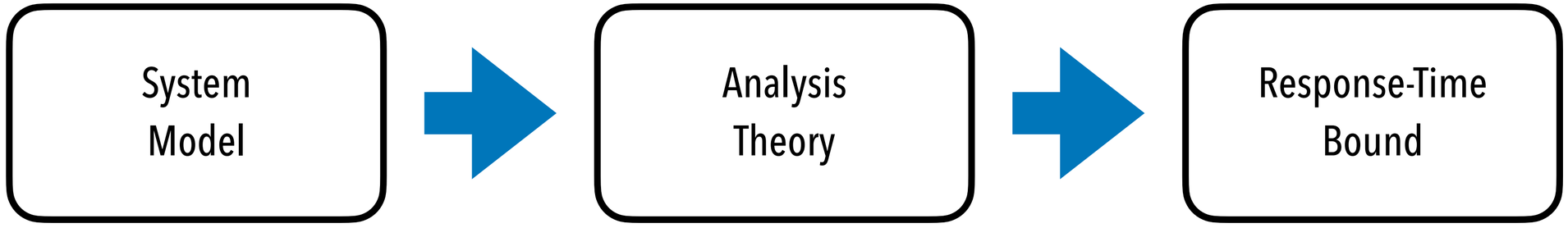
Conventional RTA yields just a number…



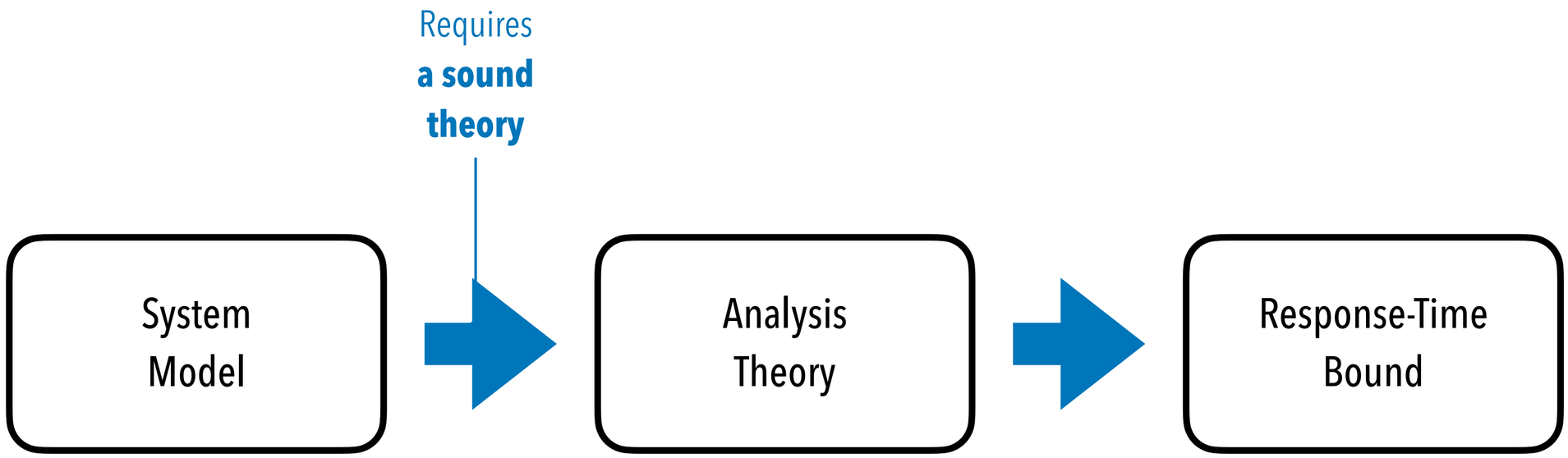
…so the evaluator must **trust** the process by which the result was obtained.

Should we?

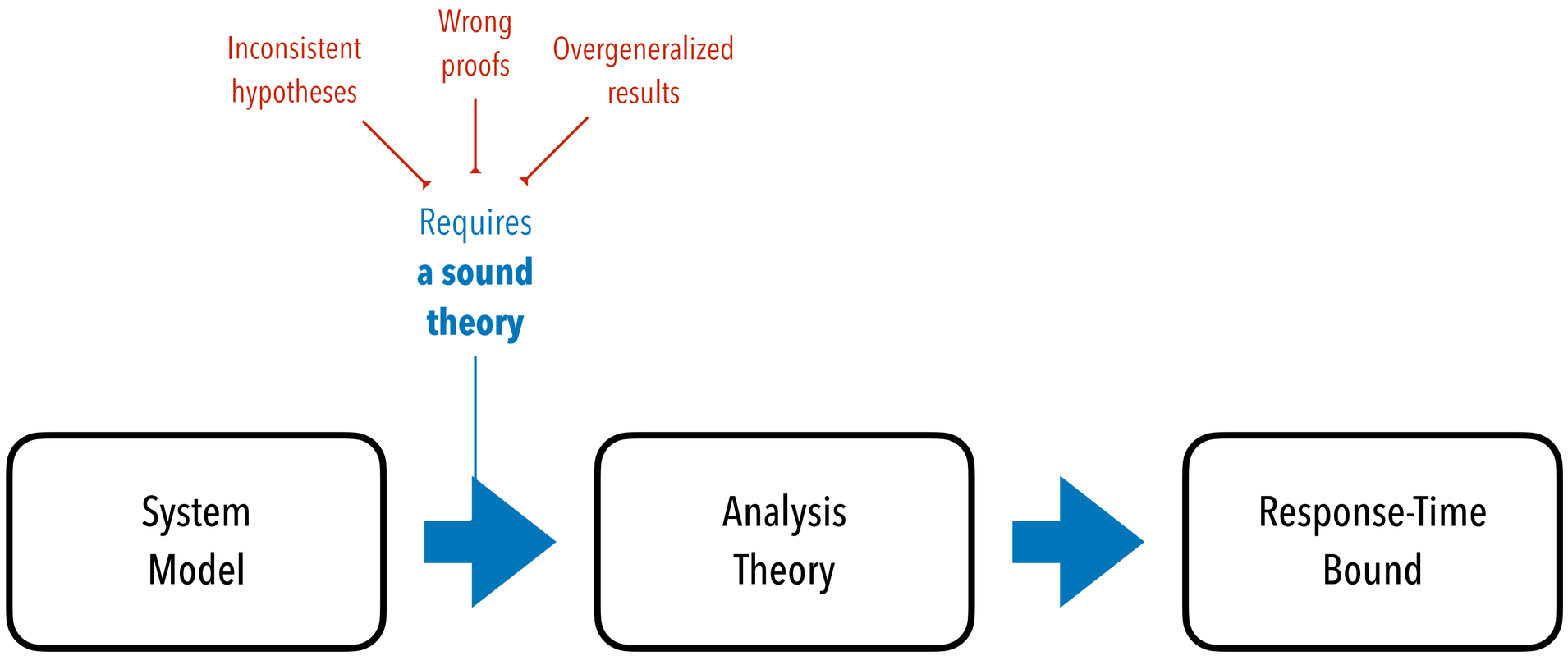
ISSUE 2: RTA IS ERROR PRONE



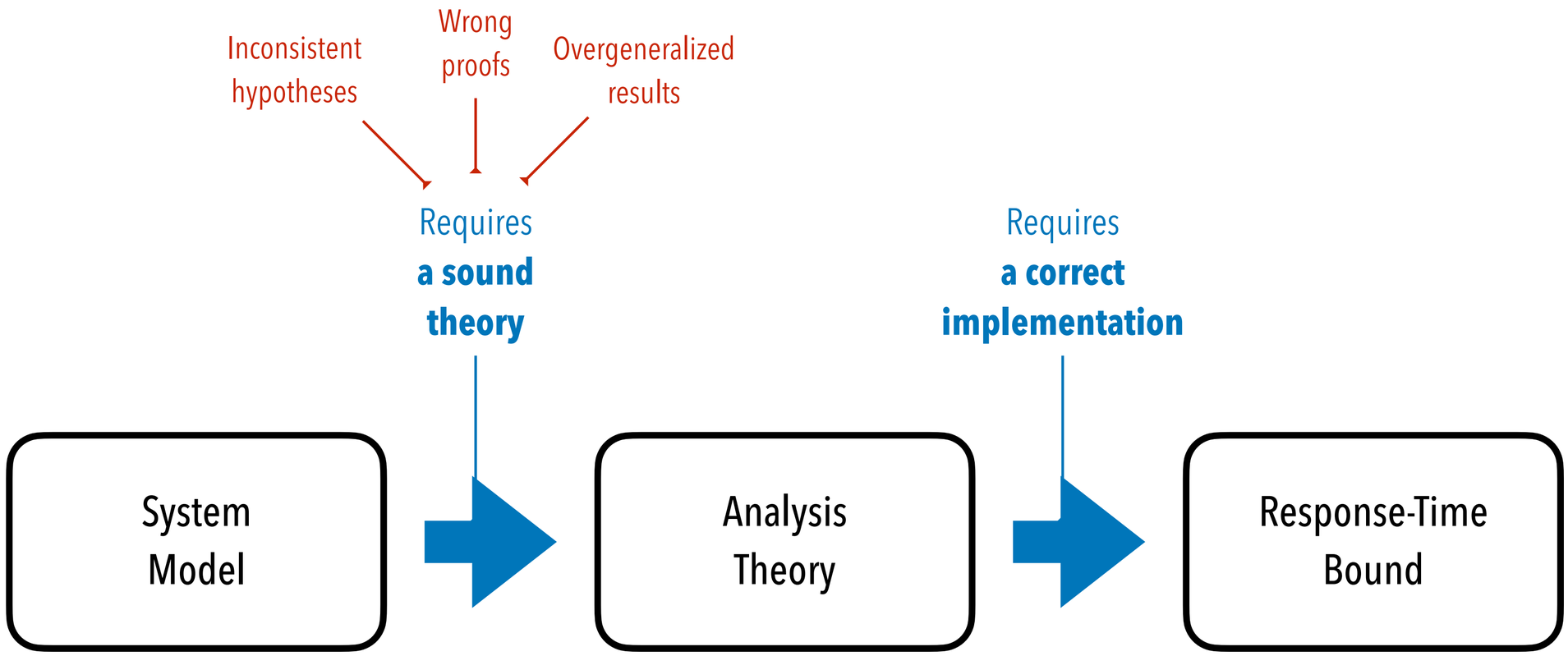
ISSUE 2: RTA IS ERROR PRONE



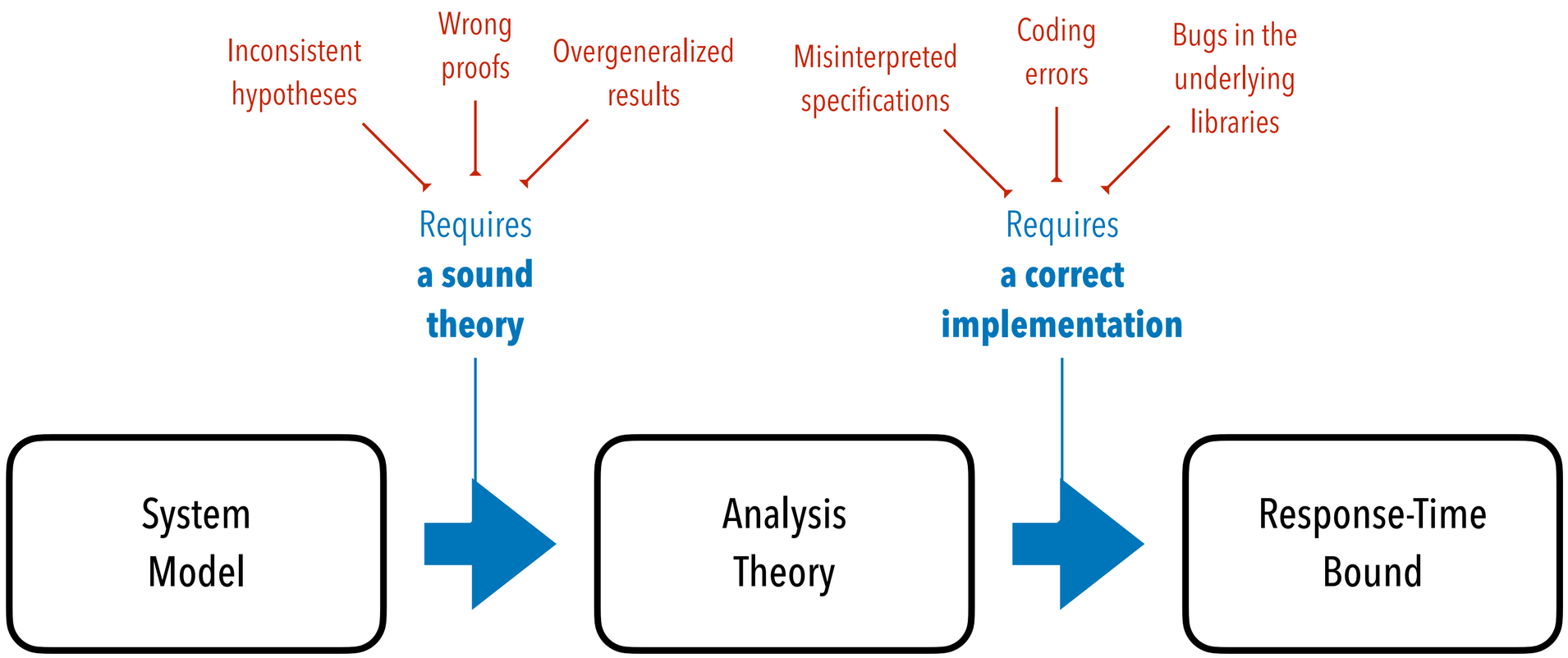
ISSUE 2: RTA IS ERROR PRONE



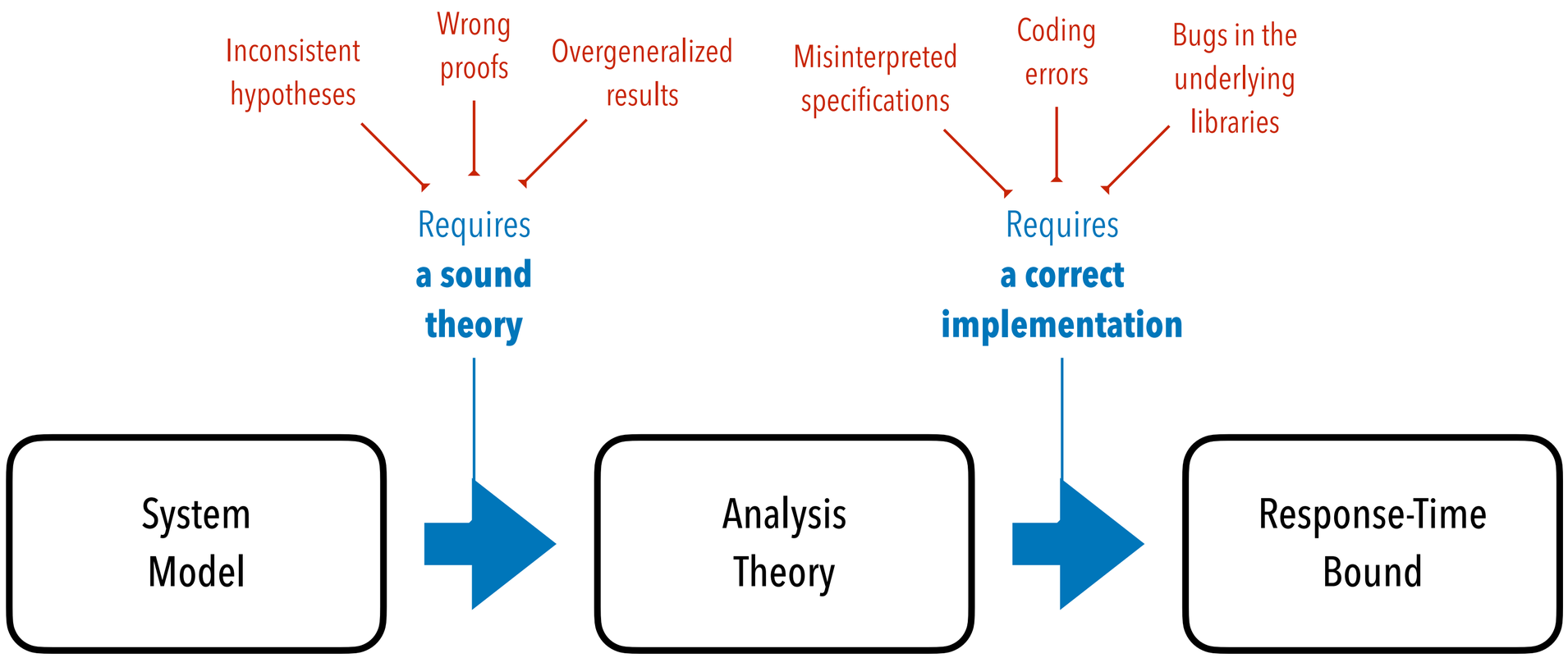
ISSUE 2: RTA IS ERROR PRONE



ISSUE 2: RTA IS ERROR PRONE



ISSUE 2: RTA IS ERROR PRONE



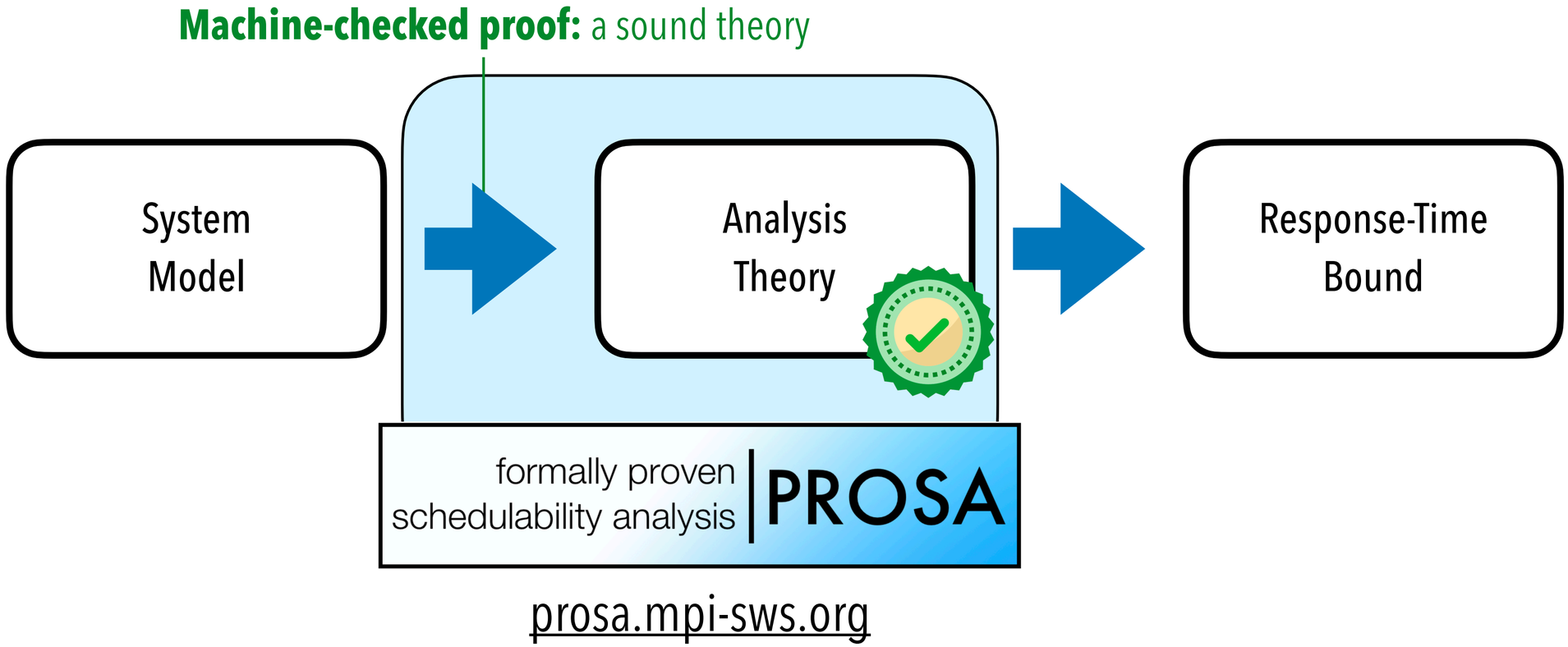
Can we make use of formal verification?

DESIGN SPACE

How to formally verify an RTA?

PRIOR WORK: VERIFY ONLY THE THEORY

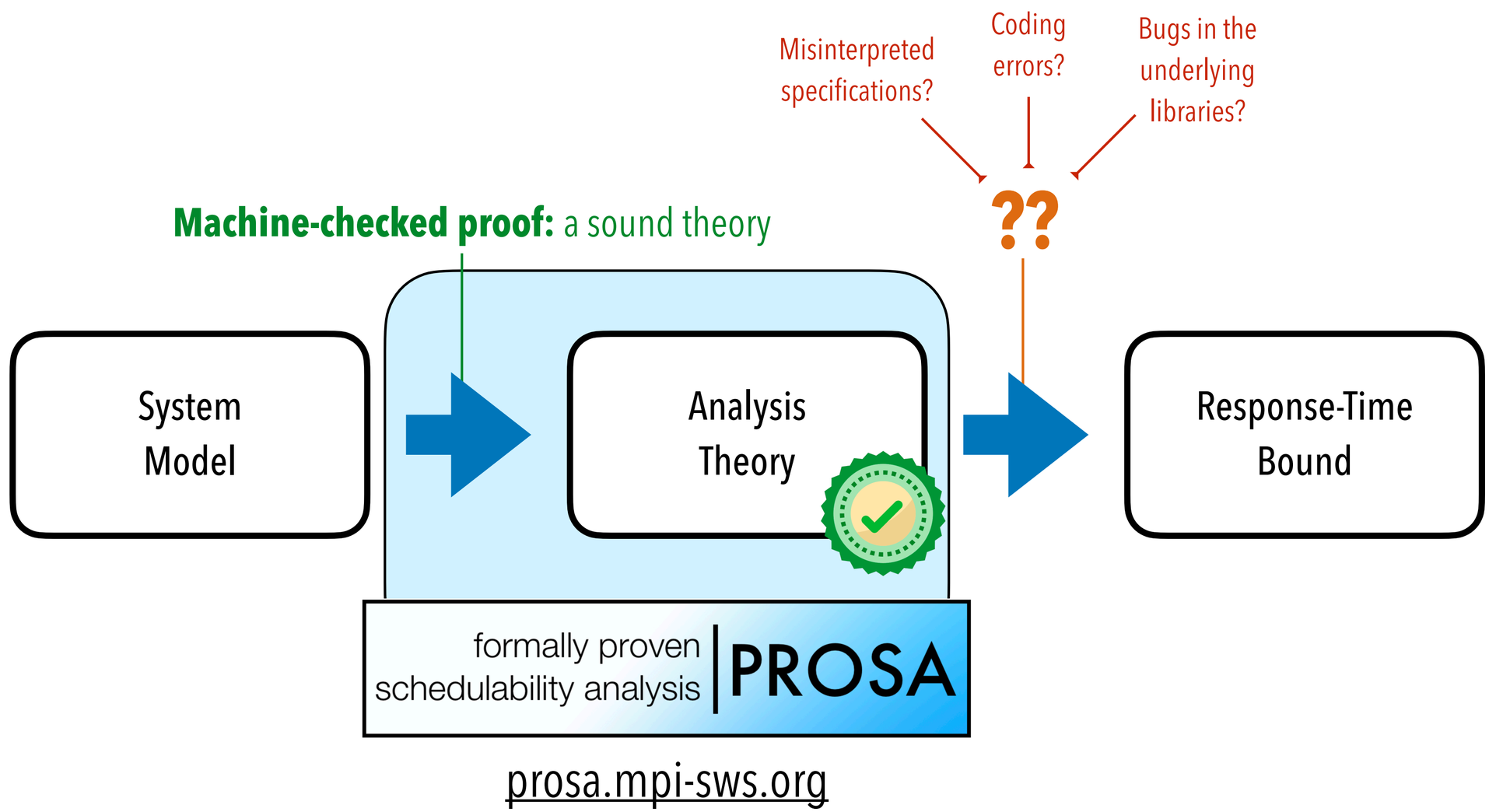
[Cerqueira et al., ECRTS 2016]



prosa.mpi-sws.org

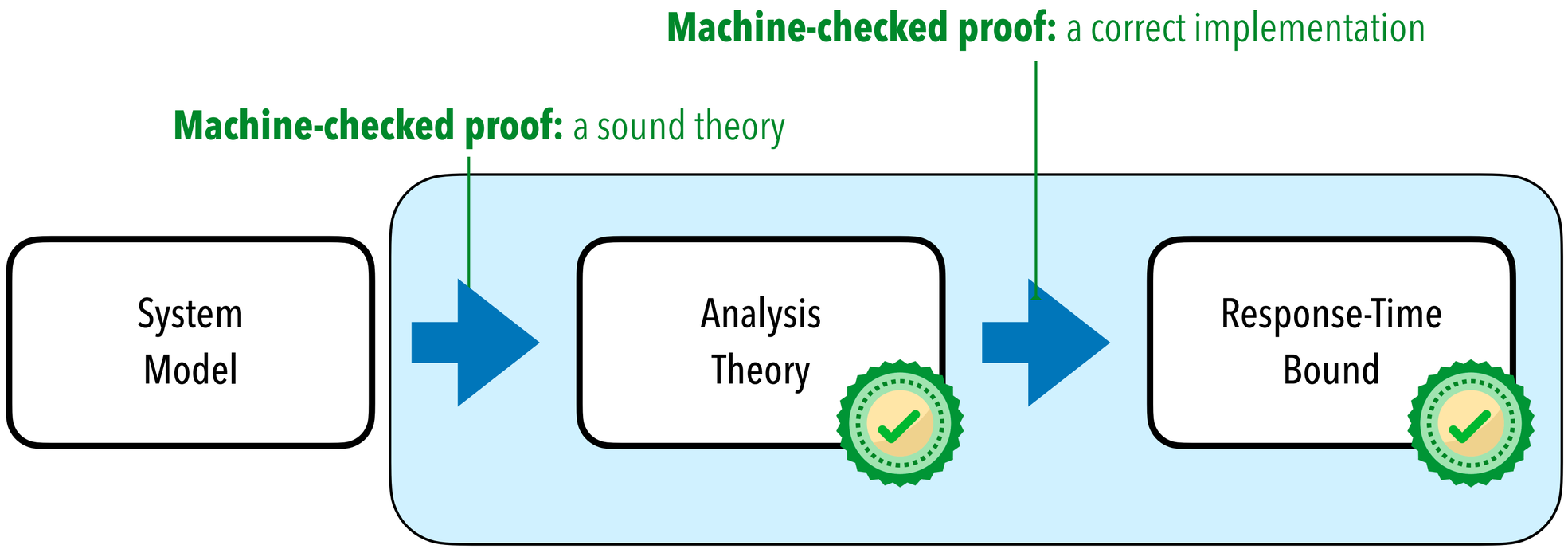
PRIOR WORK: VERIFY ONLY THE THEORY

[Cerqueira et al., ECRTS 2016]

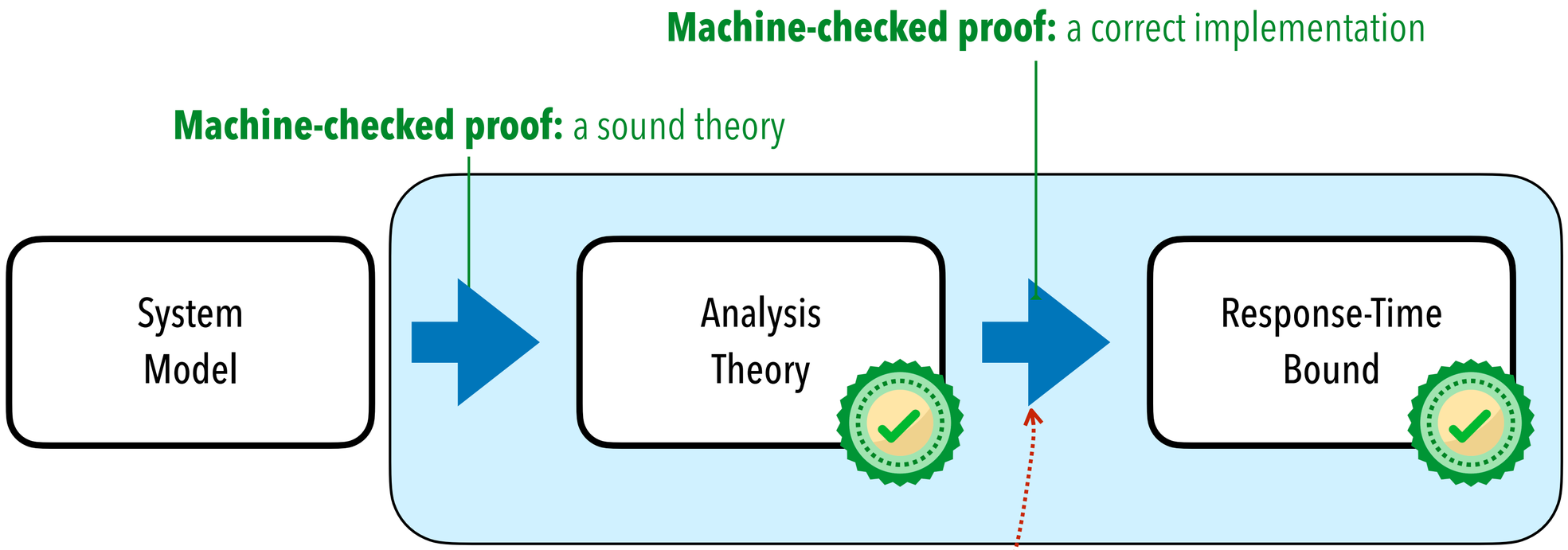


prosa.mpi-sws.org

TOUR DE FORCE: VERIFY THE ENTIRE TOOL



TOUR DE FORCE: VERIFY THE ENTIRE TOOL



A "**heroic effort**" needed:

- massive amount of work to verify *everything* (e.g., input parser, I/O, etc.)
- not easily updated once verified

TOUR DE FORCE: VERIFY THE ENTIRE TOOL

hypothetical

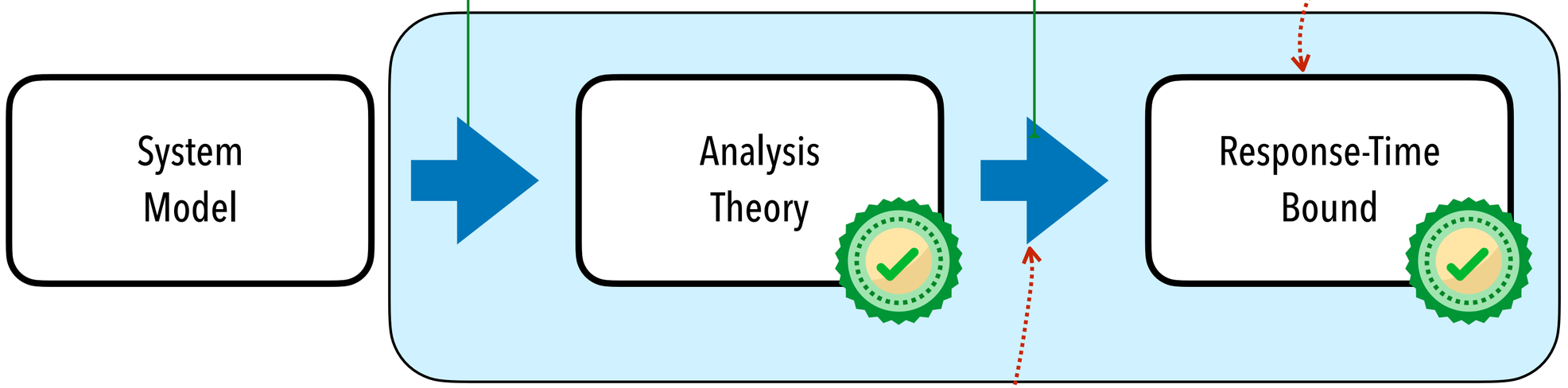
(never done in prior work)

no explainable evidence

(still just a number)

Machine-checked proof: a correct implementation

Machine-checked proof: a sound theory

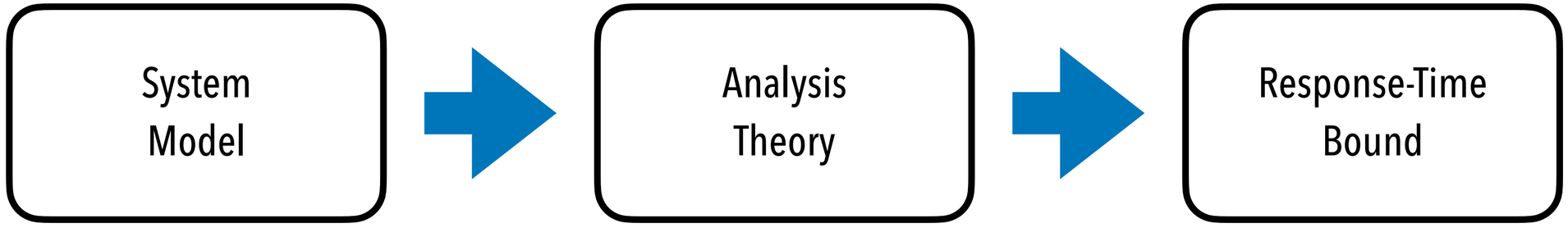


A "**heroic effort**" needed:

- massive amount of work to verify *everything* (e.g., input parser, I/O, etc.)
- not easily updated once verified

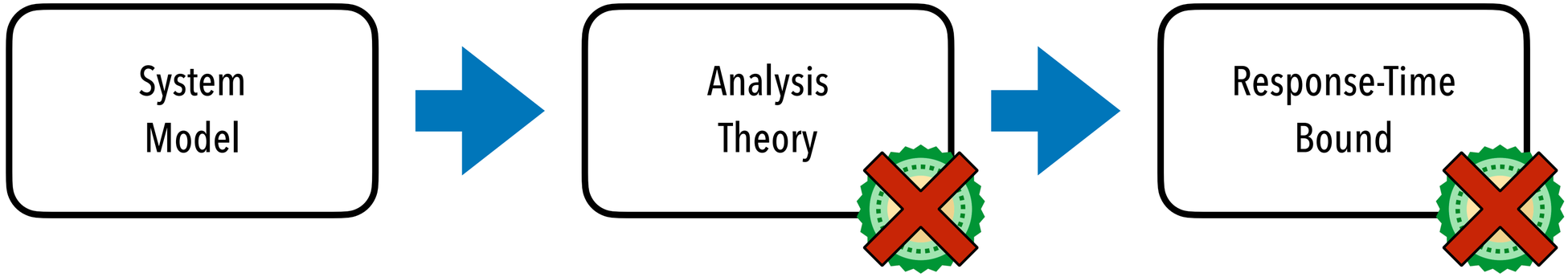
PRIOR WORK: VERIFY A RESULT VALIDATION PROCEDURE

[CertiCAN, Fradet et al., RTAS 2019]



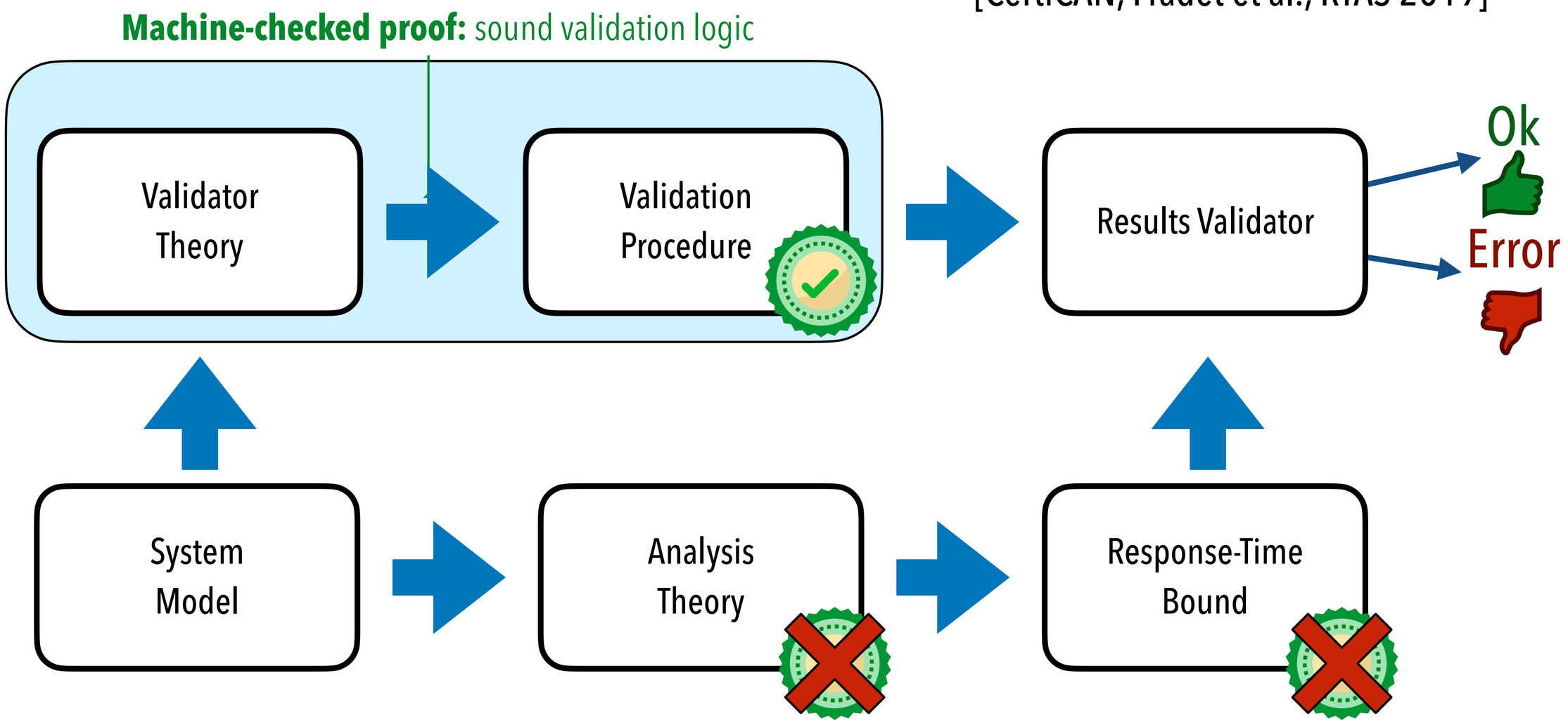
PRIOR WORK: VERIFY A RESULT VALIDATION PROCEDURE

[CertiCAN, Fradet et al., RTAS 2019]



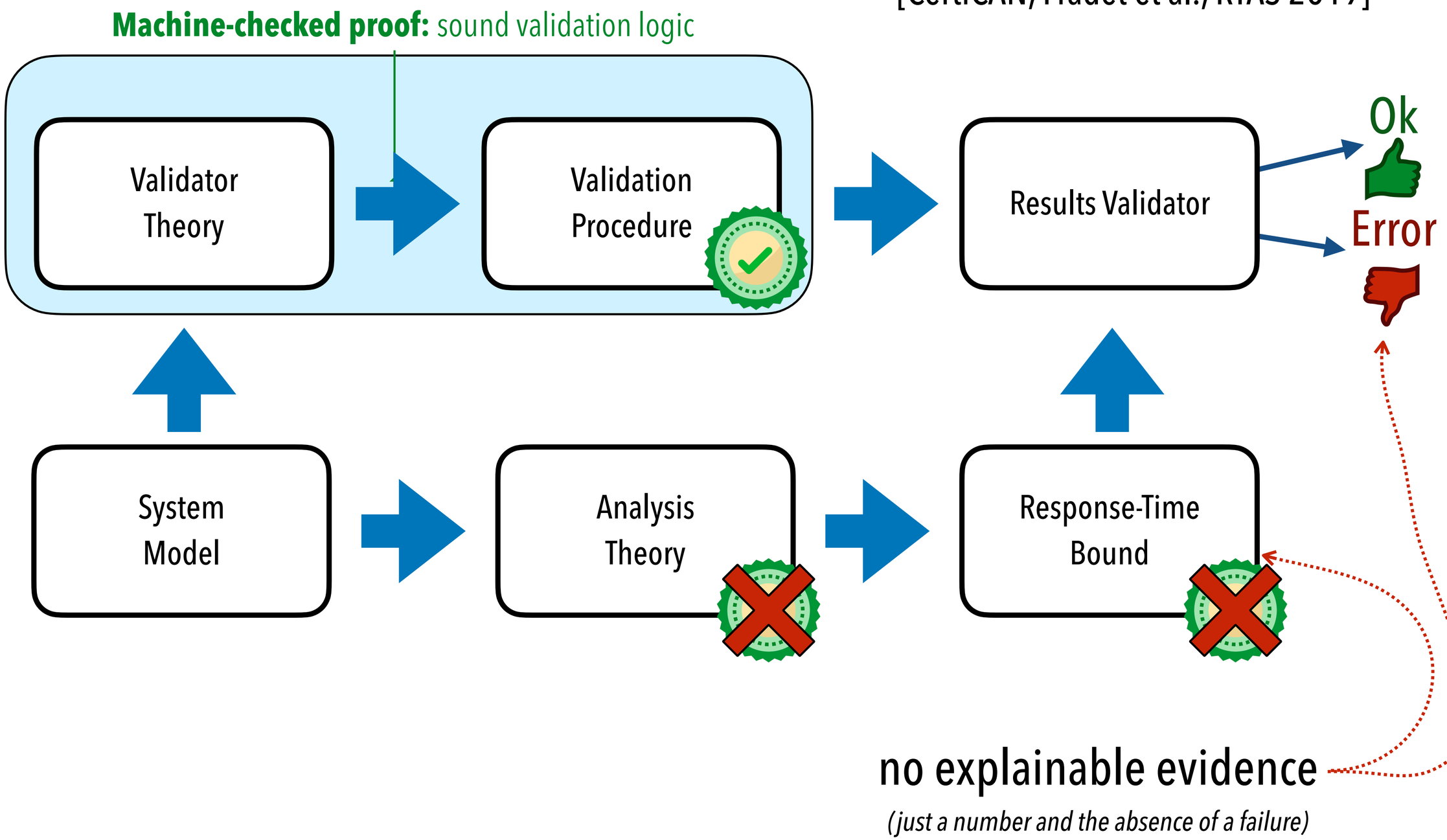
PRIOR WORK: VERIFY A RESULT VALIDATION PROCEDURE

[CertiCAN, Fradet et al., RTAS 2019]



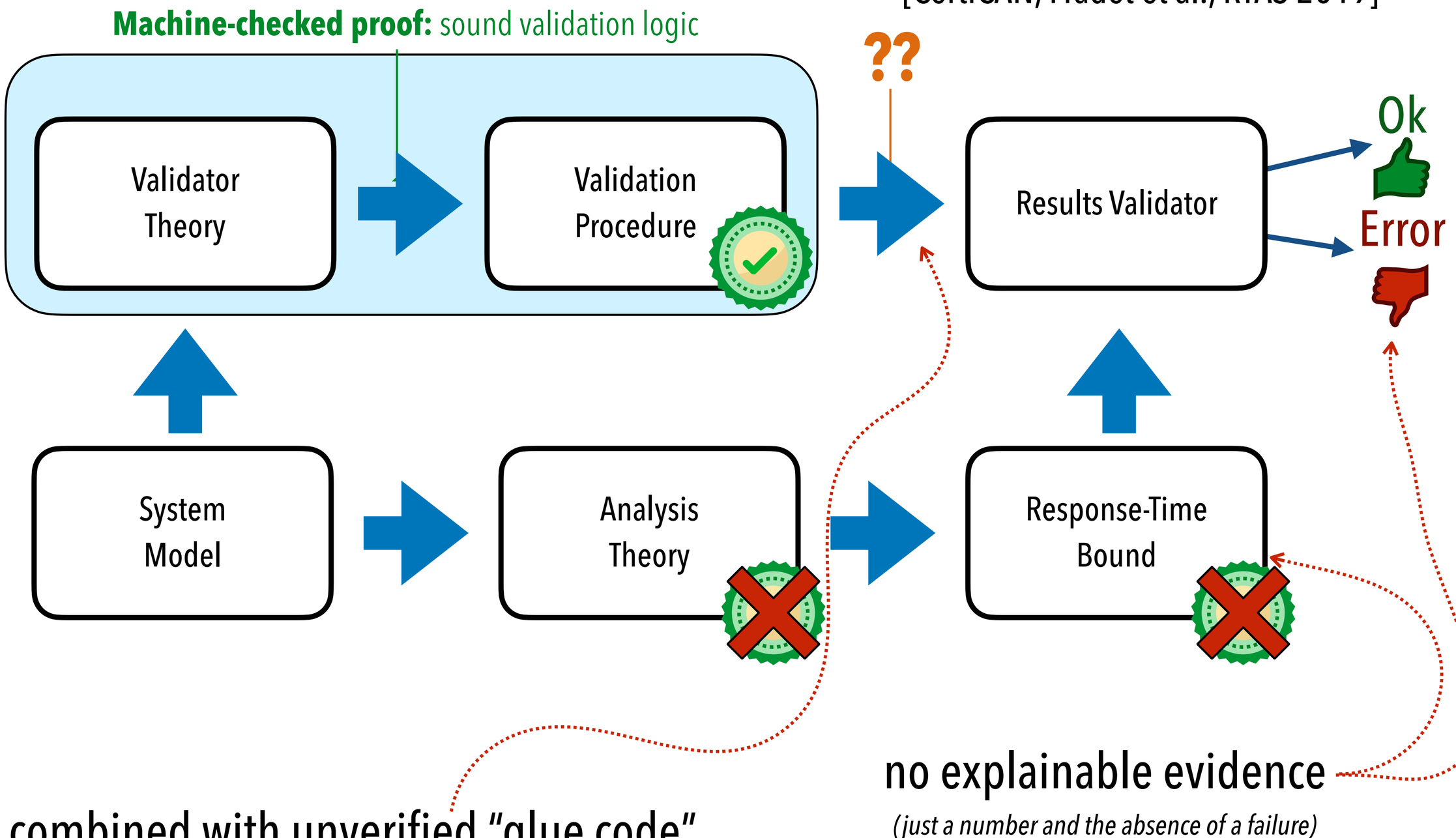
PRIOR WORK: VERIFY A RESULT VALIDATION PROCEDURE

[CertiCAN, Fradet et al., RTAS 2019]



PRIOR WORK: VERIFY A RESULT VALIDATION PROCEDURE

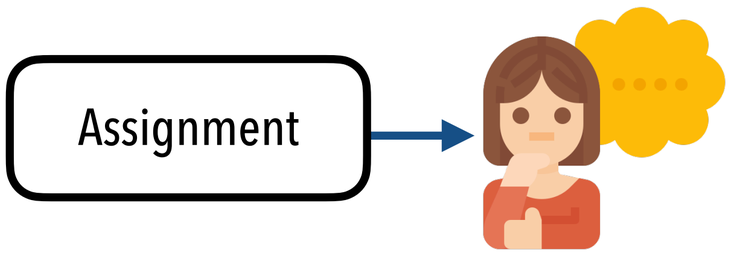
[CertiCAN, Fradet et al., RTAS 2019]



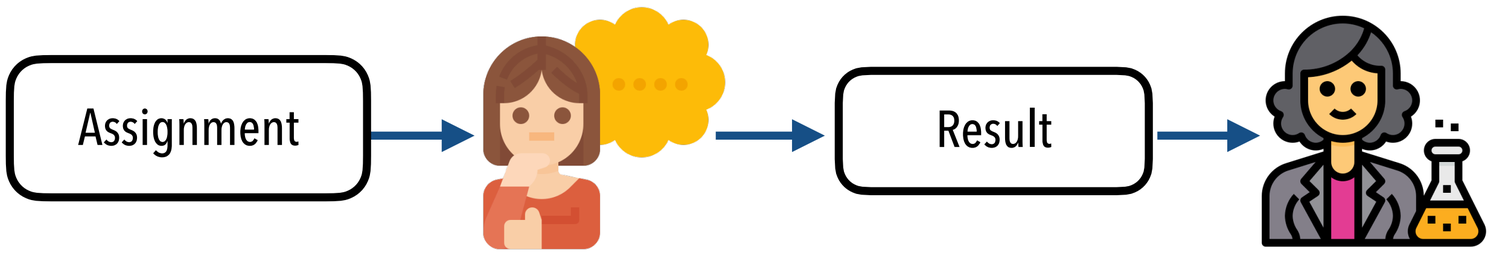
A NEW APPROACH:

FOUNDATIONAL RTA

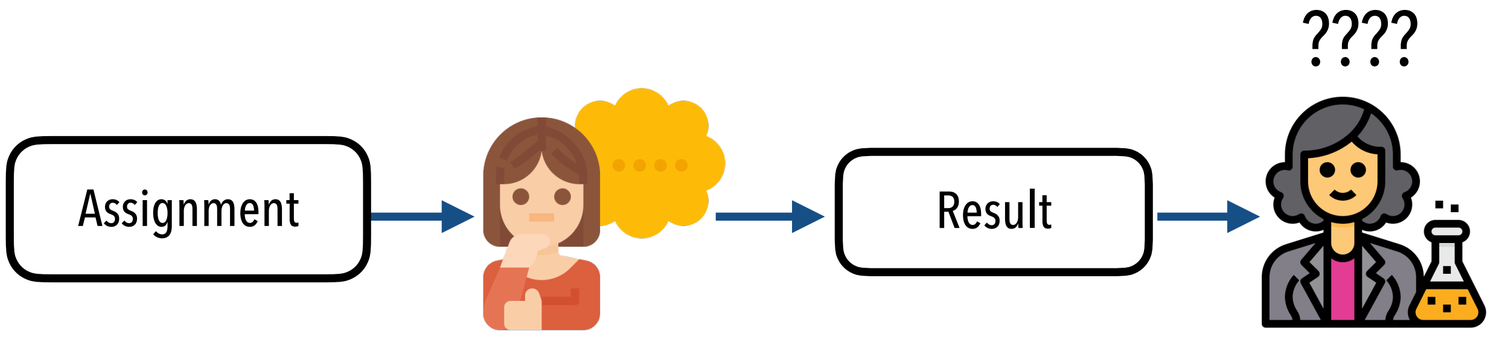
DISTANT ANALOGY: SHOW YOUR WORK



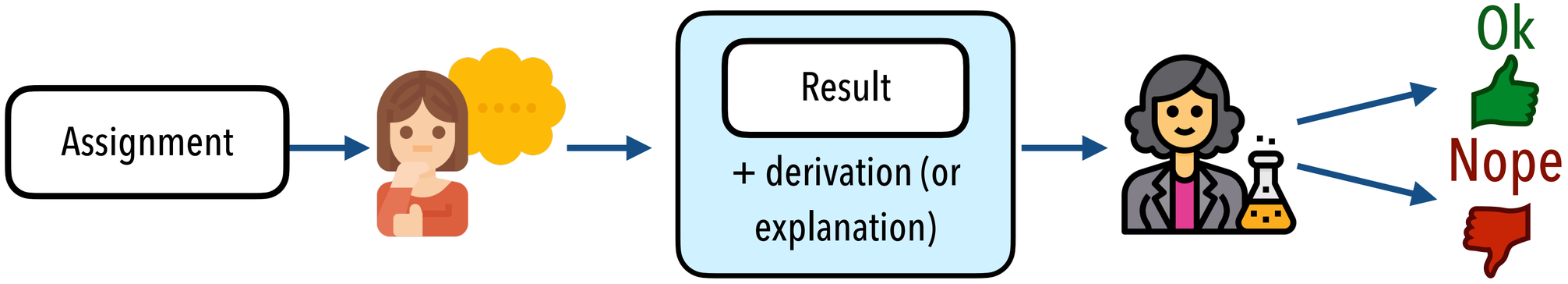
DISTANT ANALOGY: SHOW YOUR WORK



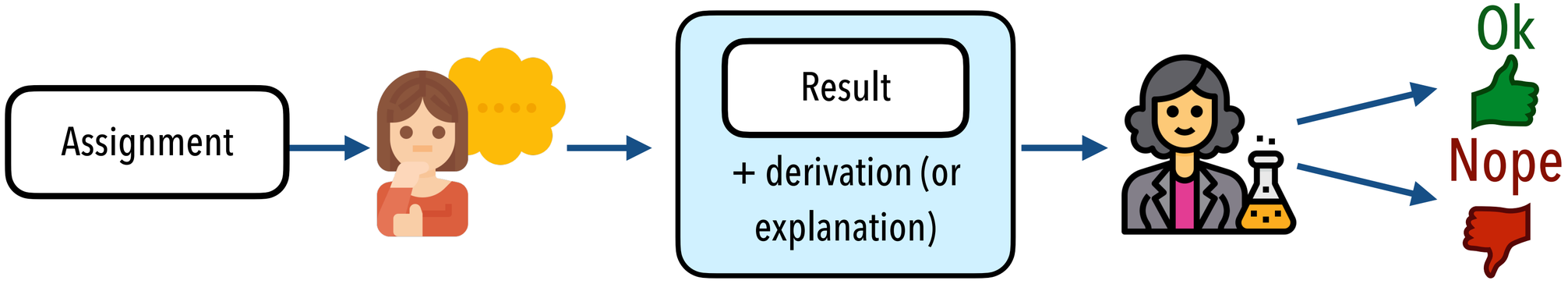
DISTANT ANALOGY: SHOW YOUR WORK



DISTANT ANALOGY: SHOW YOUR WORK



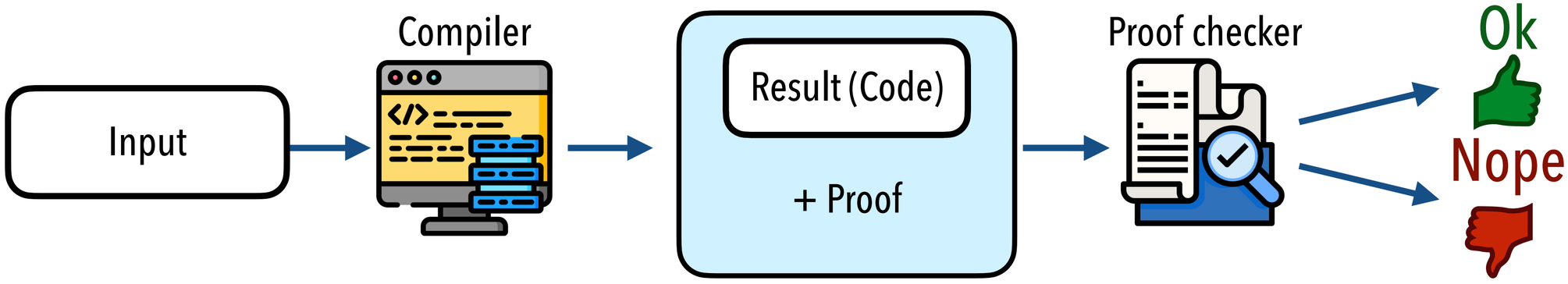
DISTANT ANALOGY: SHOW YOUR WORK



Idea: report both results and an argument for why the results are correct

PROOF-CARRYING CODE

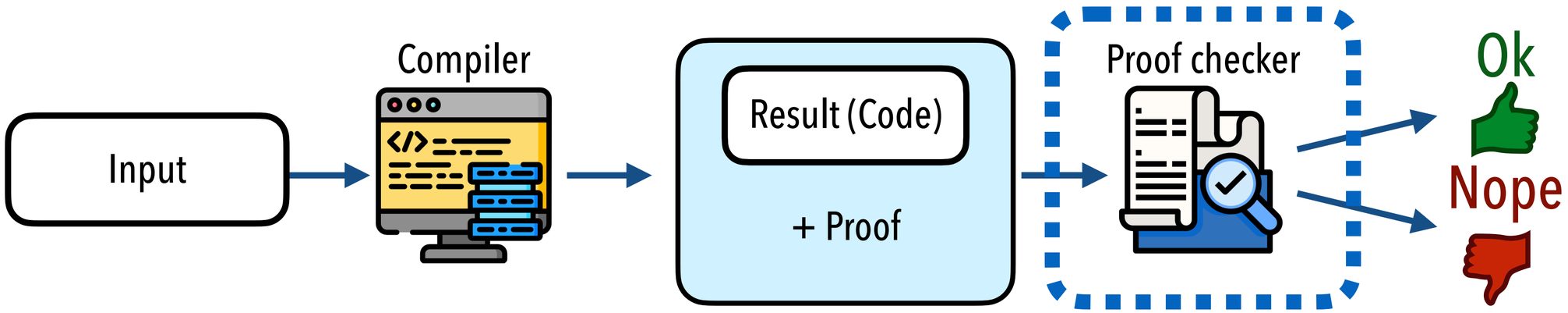
[Necula, POPL 1997]



Idea: tool must produce both *results* and *proofs* that the results are correct

PROOF-CARRYING CODE

[Necula, POPL 1997]



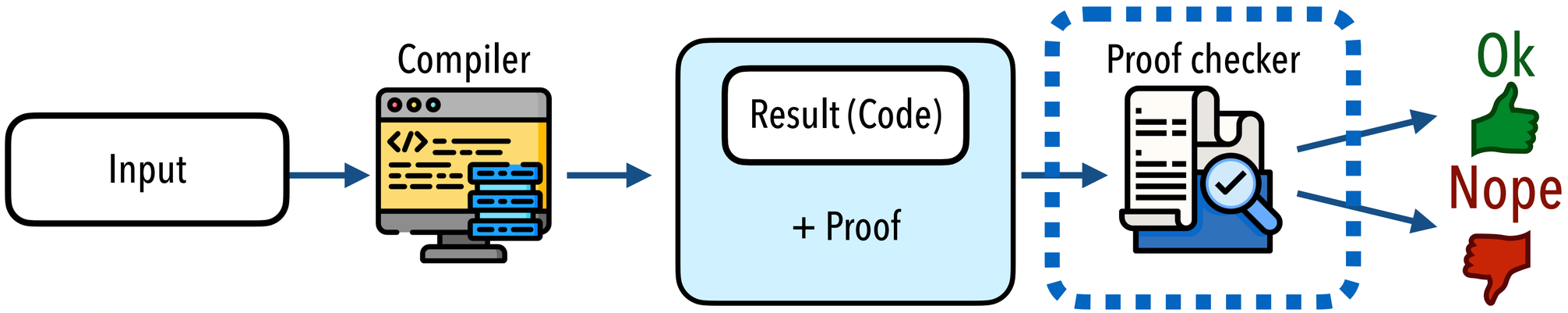
Idea: tool must produce both *results* and *proofs* that the results are correct

Trusted Computing Base (TCB)

does not include the compiler

PROOF-CARRYING CODE

[Necula, POPL 1997]



Idea: tool must produce both *results* and *proofs* that the results are correct

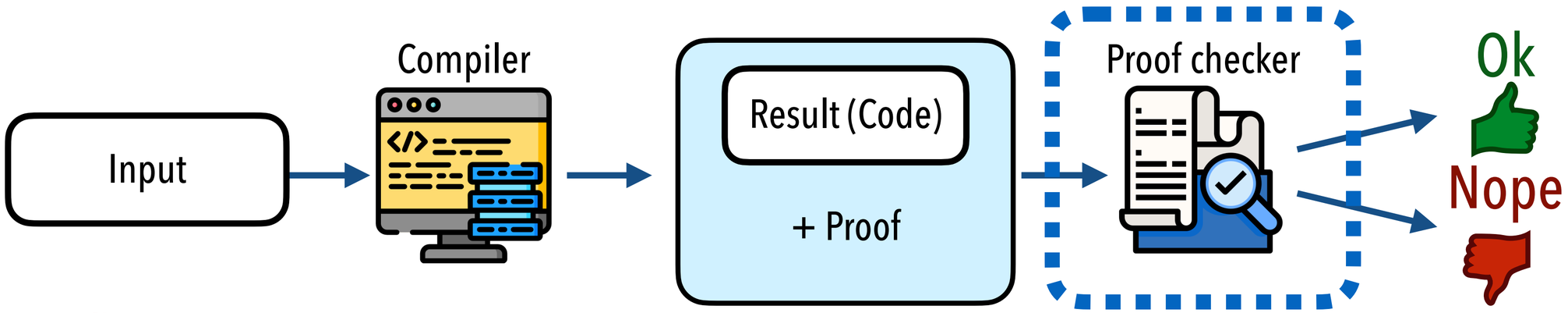
But who is checking the checker?

Trusted Computing Base (TCB)

does not include the compiler

PROOF-CARRYING CODE

[Necula, POPL 1997]



Idea: tool must produce both *results* and *proofs* that the results are correct

Trusted Computing Base (TCB)

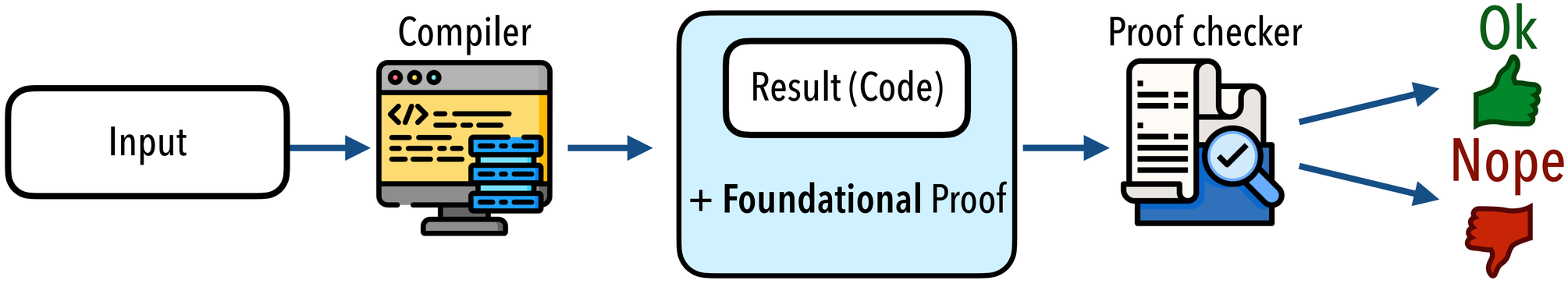
does not include the compiler

But who is checking the checker?

➔ Keep the checker as **simple** as possible!

FOUNDATIONAL PROOF-CARRYING CODE

[Appel, LICS 2001]



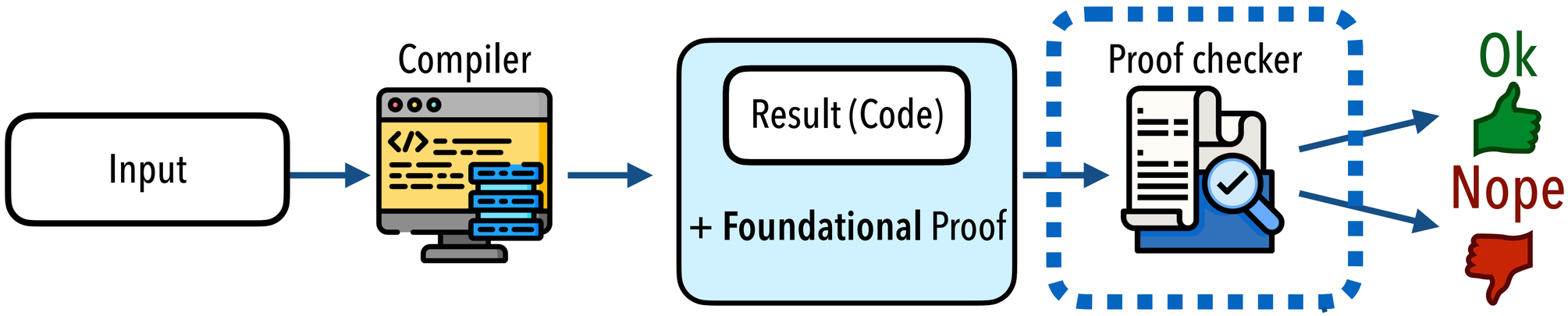
Idea: tool must produce both *results* and a *foundational proof* of correctness

Foundational proof

= proof relies only the foundations of mathematical logic

FOUNDATIONAL PROOF-CARRYING CODE

[Appel, LICS 2001]



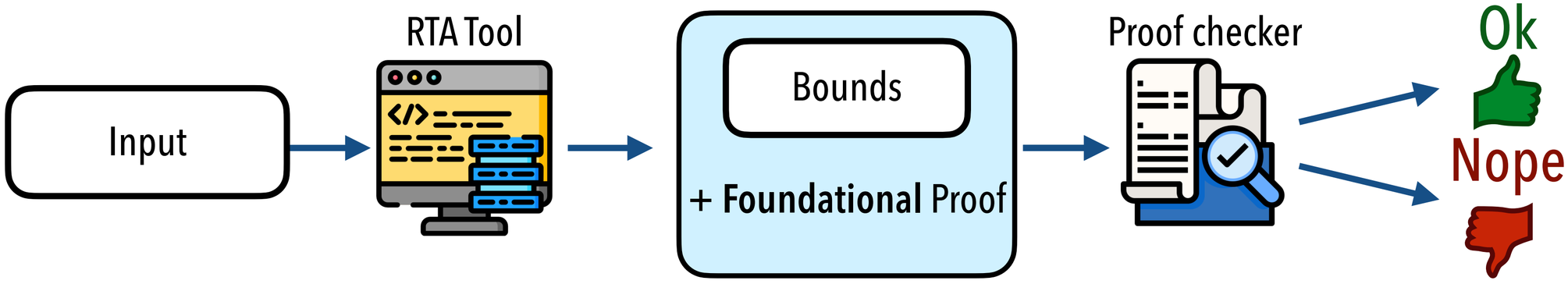
Idea: tool must produce both *results* and a *foundational proof* of correctness

Goal: minimal TCB

Foundational proof

= proof relies only the foundations of mathematical logic

THIS PAPER: PROOF-CARRYING RESPONSE-TIME BOUNDS



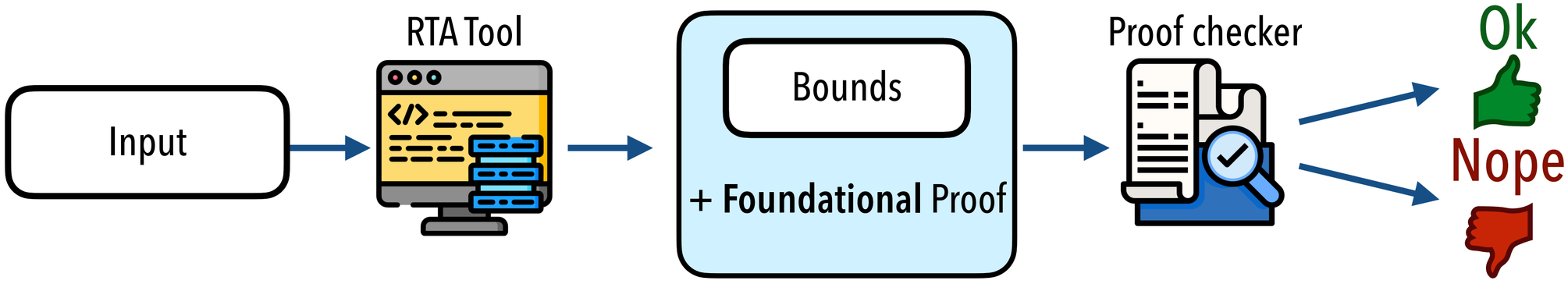
Idea: RTA must produce both *results* and a *foundational proof* of correctness

➔ Provably **safe** results

➔ **Small TCB**, not including RTA tool or theory

➔ Independently checkable **evidence** of timeliness

THIS PAPER: PROOF-CARRYING RESPONSE-TIME BOUNDS



Idea: RTA must produce both *results* and a *foundational proof* of correctness

➔ Provably **safe** results

➔ **Small TCB**, not including RTA tool or theory

➔ Independently checkable **evidence** of timeliness

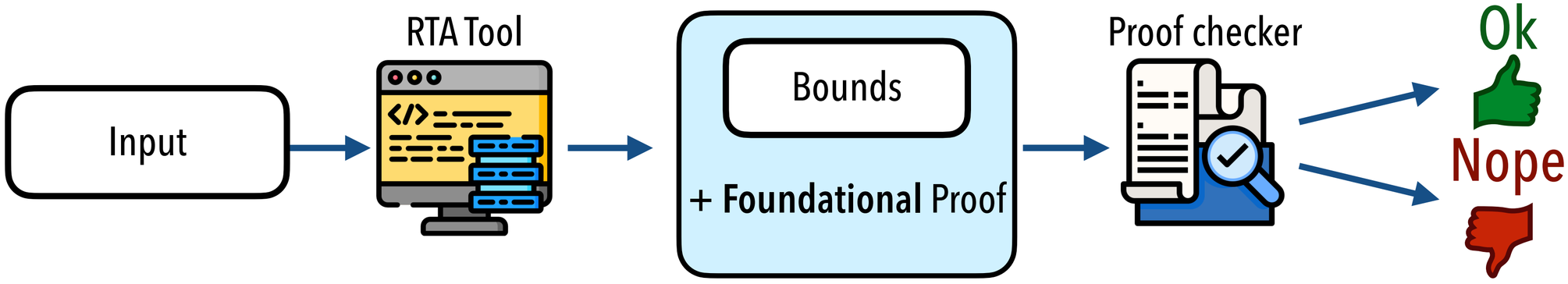
…but does it scale?

What logic to use as a foundation?

How to automatically generate proofs?

How to keep proofs explainable?

THIS PAPER: PROOF-CARRYING RESPONSE-TIME BOUNDS



Idea: RTA must produce both *results* and a *foundational proof* of correctness

➔ Provably **safe** results

➔ **Small TCB**, not including RTA tool or theory

➔ Independently checkable **evidence** of timeliness

…but does it scale?

What logic to use as a foundation?

How to automatically generate proofs?

How to keep proofs explainable?

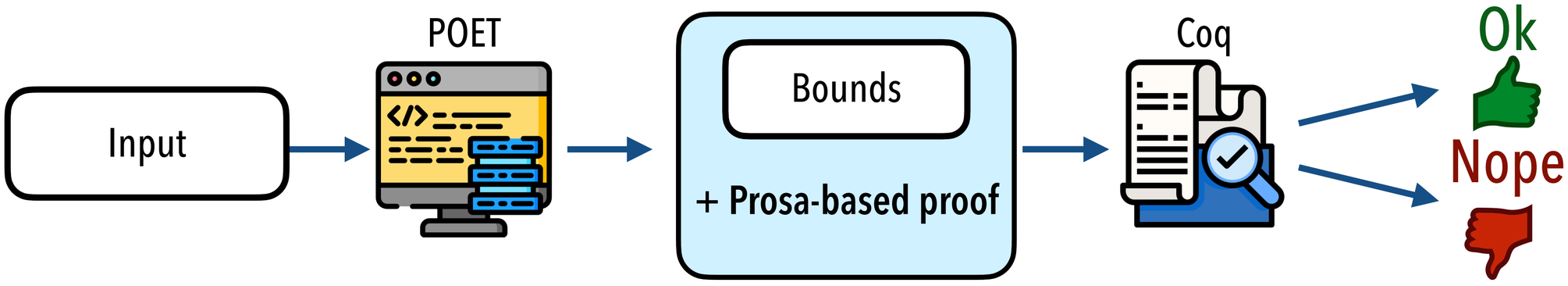
Nice concept… but does it actually work?

POET

Prosa Obsigned Evidence of Timeliness

POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



…let's have a quick look at Coq and Prosa.

BACKGROUND: COQ

- Coq is a **proof assistant**
- It is used to write **programs**/**definitions** and to **prove theorems**
- The **proof engine** is **not** fully automatic!

```
Theorem a_simple_theorem:
  ∀ x y,
  x + y = y + x.
Proof.
  move⇒ x y.
  induction x.
  - by rewrite add0n addn0.      (* base *)
  - by rewrite addSn IHx addnS. (* step *)
Qed.
```

BACKGROUND: COQ

- Coq is a **proof assistant**
- It is used to write **programs**/**definitions** and to **prove theorems**
- The **proof engine** is **not** fully automatic!

We state theorems

We prove theorems

```
Theorem a_simple_theorem:
  ∀ x y,
  x + y = y + x.
Proof.
  move⇒ x y.
  induction x.
  - by rewrite add0n addn0.      (* base *)
  - by rewrite addSn IHx addnS. (* step *)
Qed.
```

BACKGROUND: COQ

- Coq is a **proof assistant**
- It is used to write **programs**/**definitions** and to **prove theorems**
- The **proof engine** is **not** fully automatic!

We state theorems

We prove theorems

```
Theorem a_simple_theorem:
  ∀ x y,
  x + y = y + x.
Proof.
  move⇒ x y.
  induction x.
  - by rewrite add0n addn0.      (* base *)
  - by rewrite addSn IHx addnS. (* step *)
Qed.
```

Coq **checks** that proofs are correct

BACKGROUND:PROSA

Prosa is the to-date largest

machine-checked **framework** for **Real Time Systems Theory**

```
Variables (j : Job) (t1 : instant) (δ : duration).

(* ... *)

Lemma busy_interval_is_bounded:
  ∃ (t2 : instant),
    t2 ≤ t1 + δ ∧
    busy_interval j t1 t2.
```

formally proven schedulability analysis | PROSA

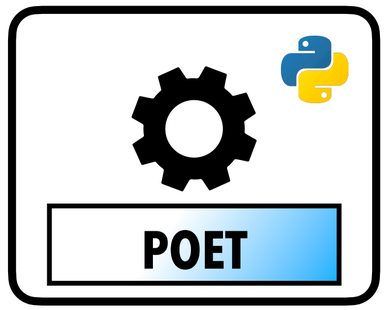
prosa.mpi-sws.org

[Cerqueira et al., ECRTS 2016]

POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq

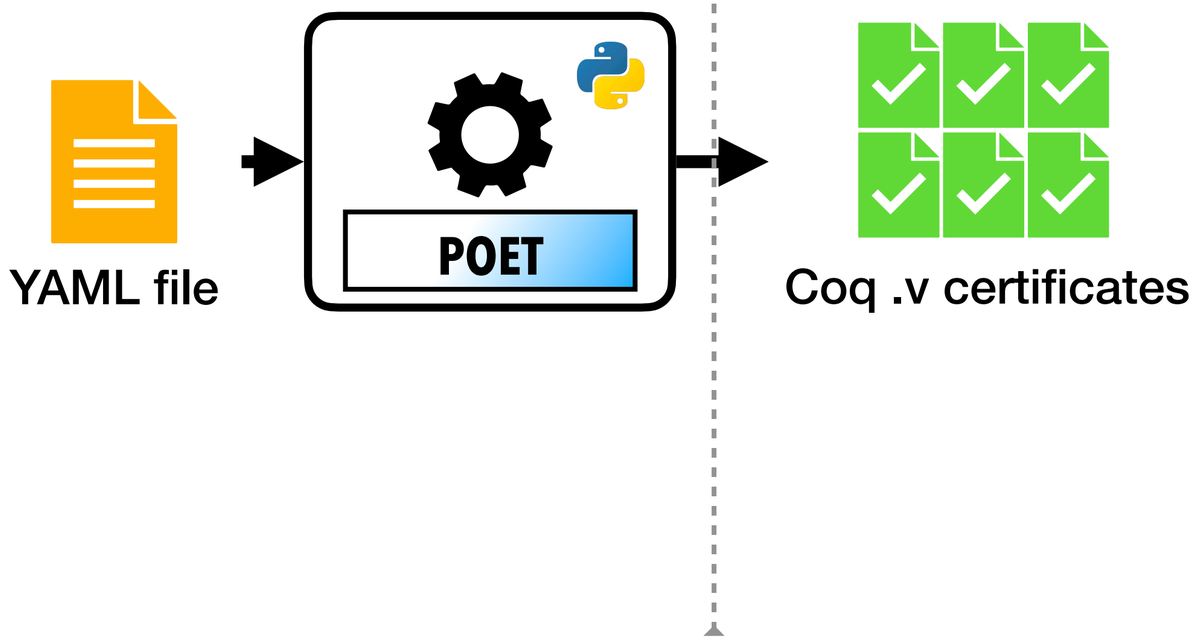
(a) Certificate generation



POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq

(a) Certificate generation

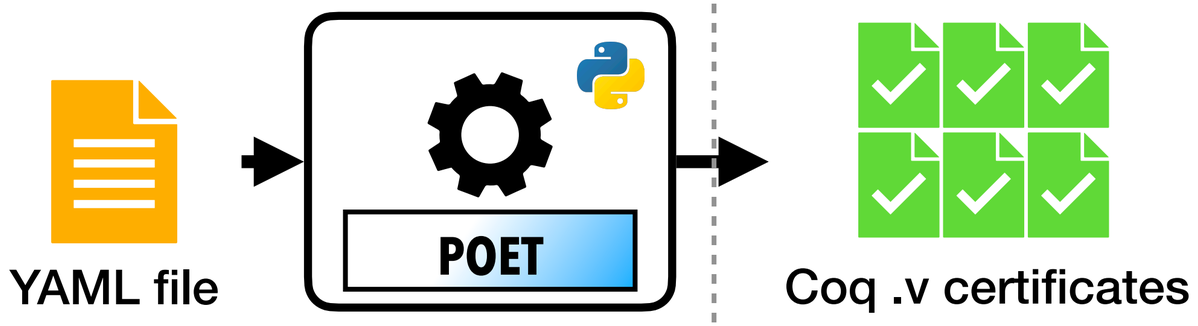


POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq

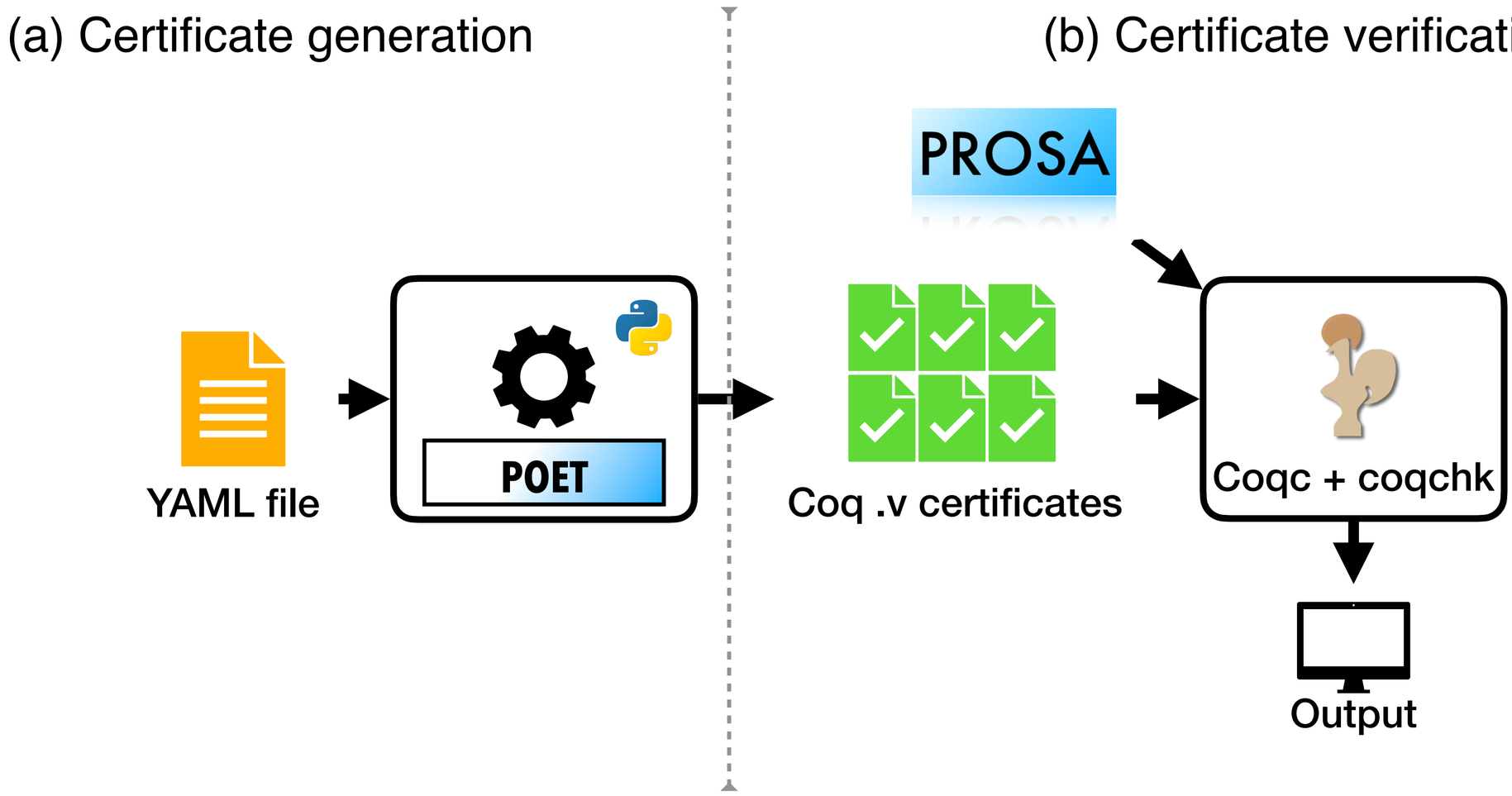
(a) Certificate generation

(b) Certificate verification



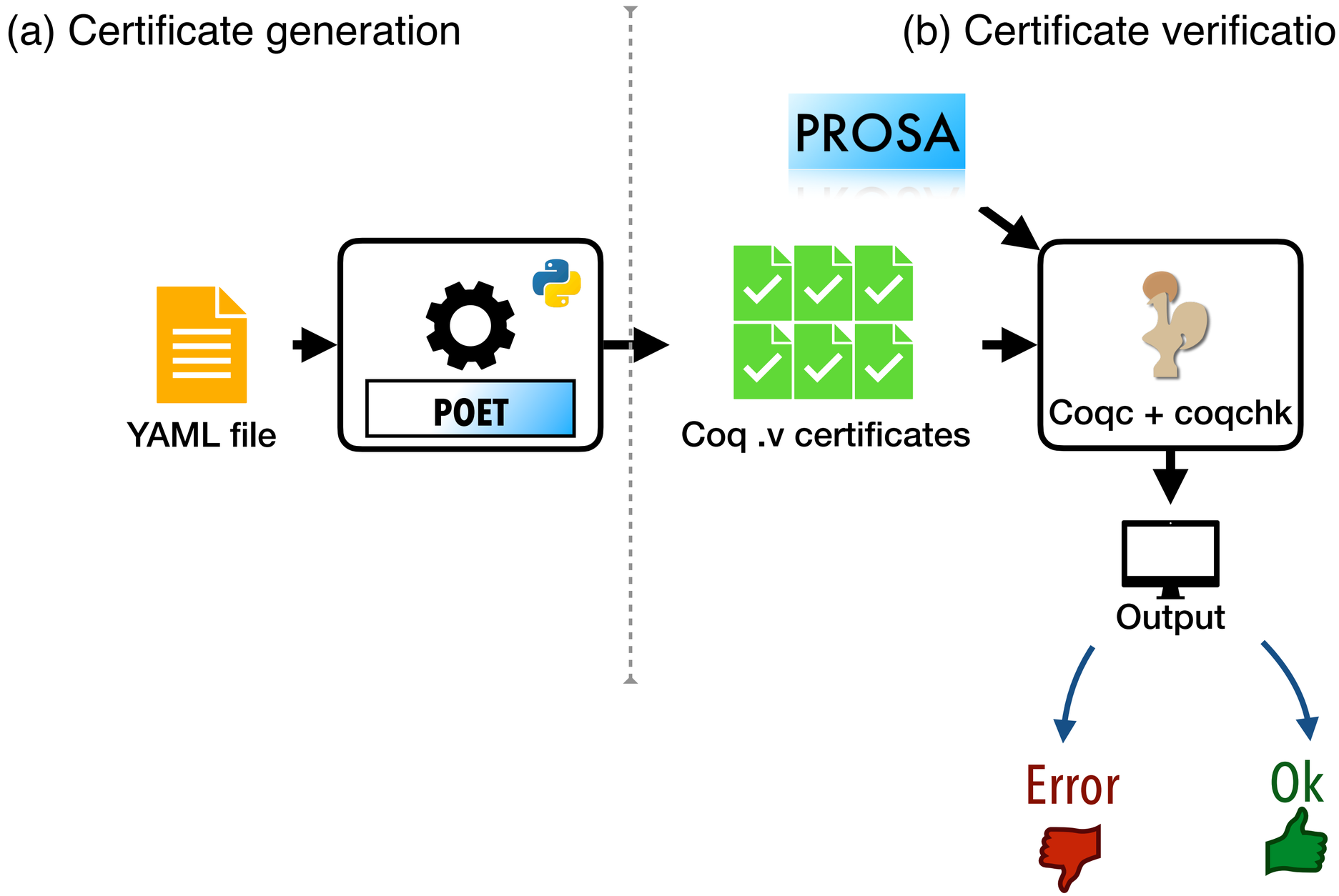
POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



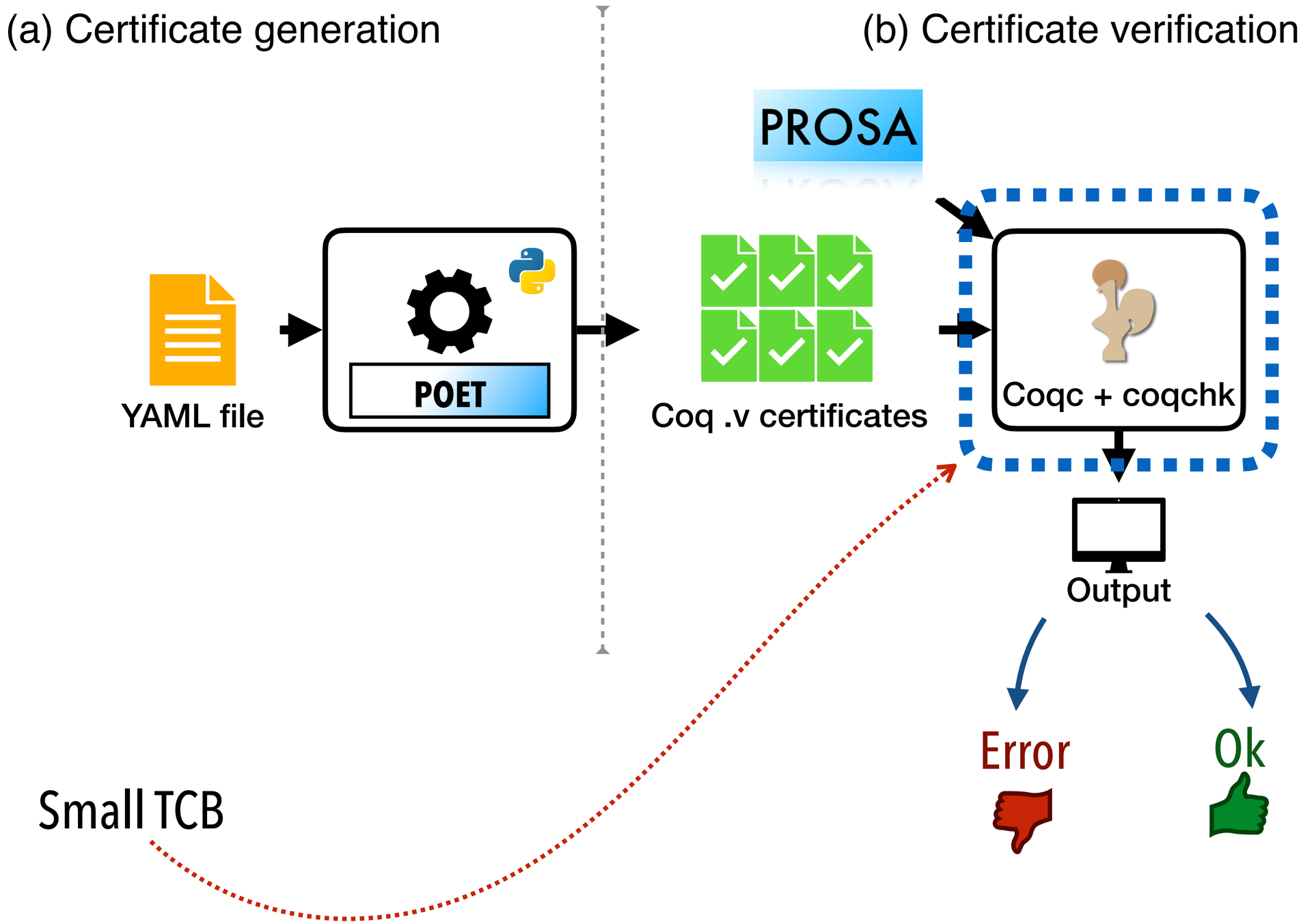
POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



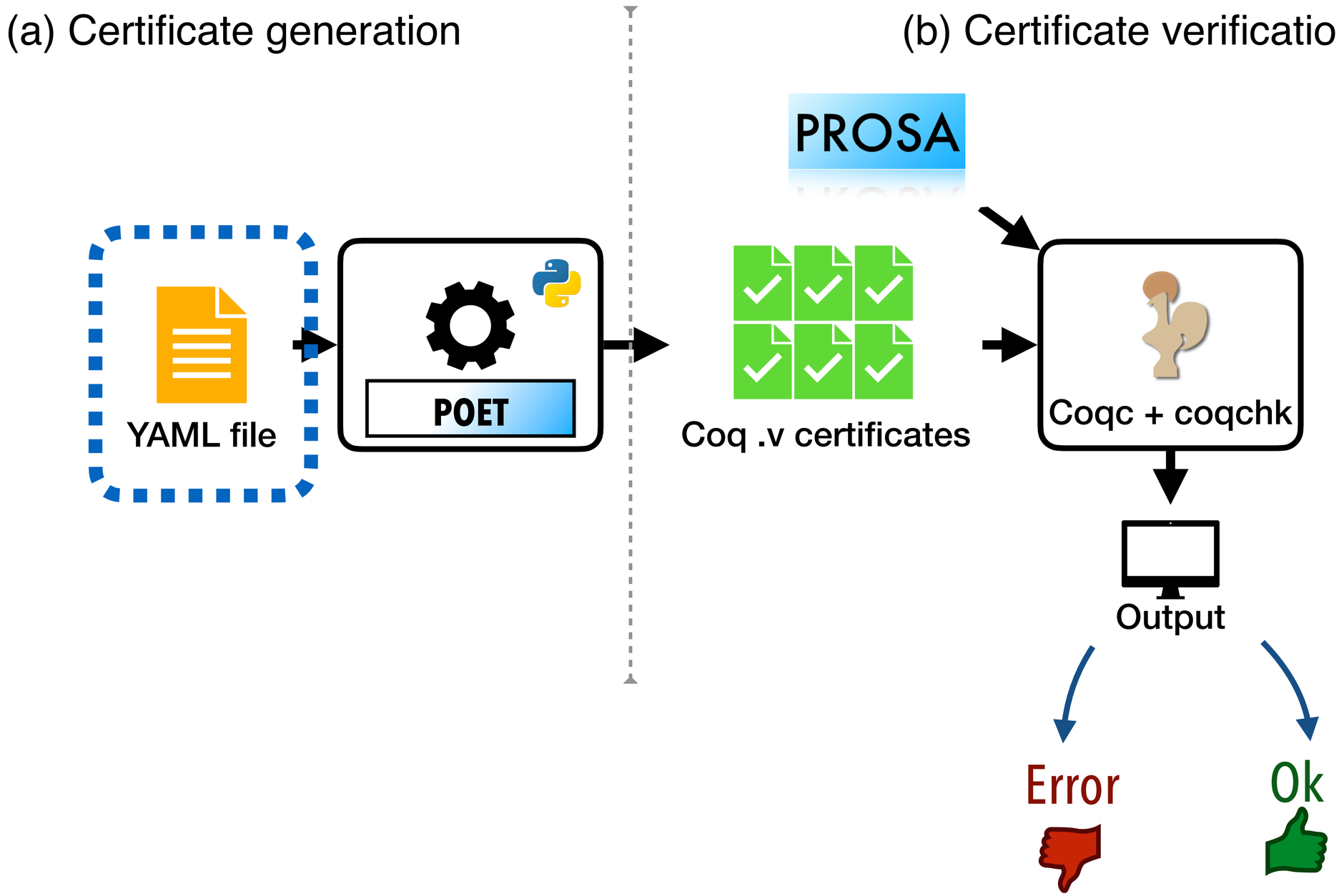
POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



INPUT FILE

YAML File

≅

Policy {

```
scheduling policy: fixed-priority
preemption policy: fully-preemptive
```

INPUT FILE

YAML File

≃

Policy

Task 1

```
scheduling policy: fixed-priority
preemption policy: fully-preemptive

- id: 1
  worst-case execution time: 50
  period: 30
  deadline: 100
  priority: 2
```

INPUT FILE

YAML File

≃

Policy

Task 1

Task 2

```
scheduling policy: fixed-priority
preemption policy: fully-preemptive

- id: 1
  worst-case execution time: 50
  period: 30
  deadline: 100
  priority: 2

- id: 2
  worst-case execution time: 10
  arrival curve: [220,[[1,1],[105,2]]]
  deadline: 100
  priority: 1
```

INPUT FILE

YAML File

≃

Policy

Task 1

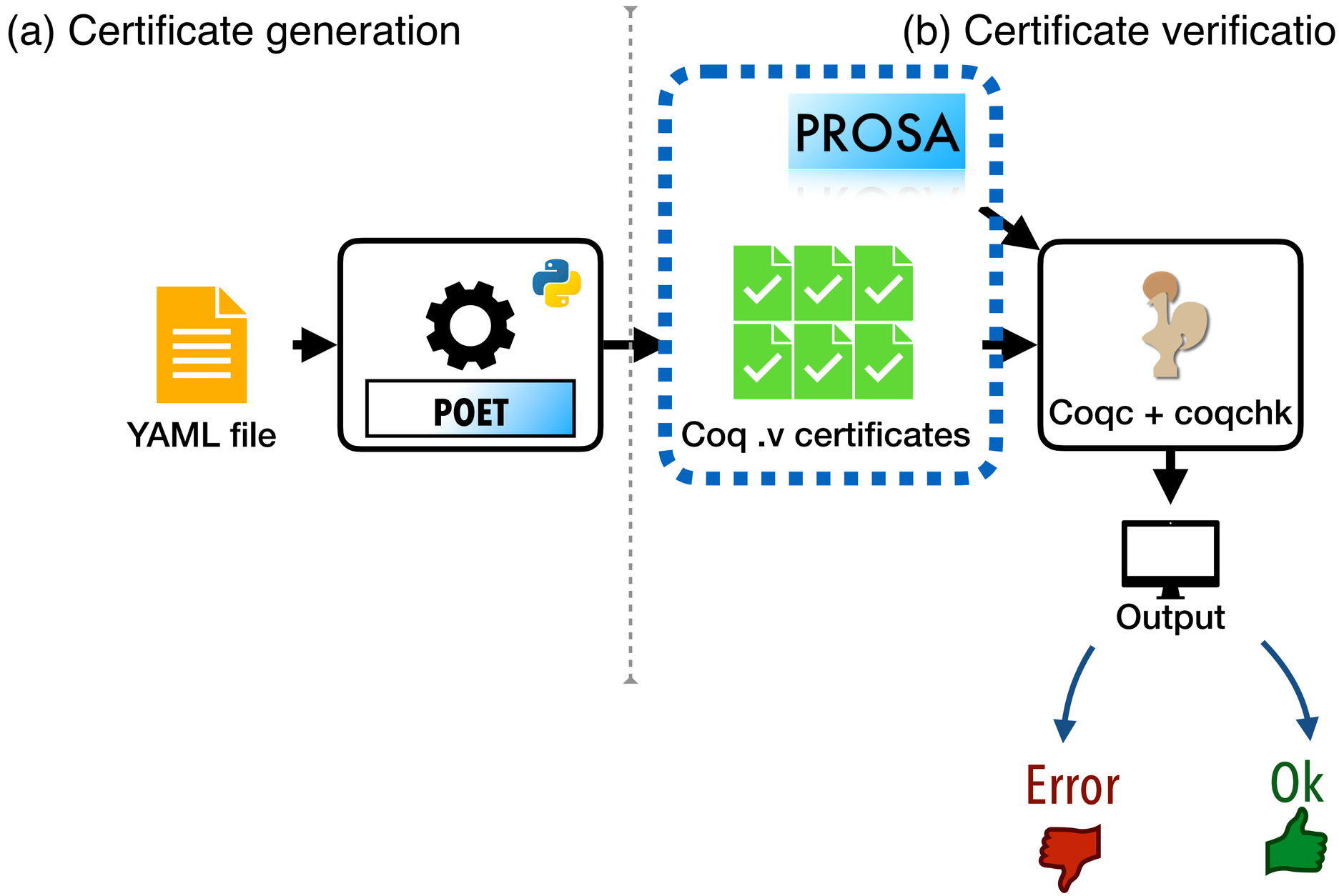
Task 2

```
scheduling policy: fixed-priority
preemption policy: fully-preemptive

- id: 1
  worst-case execution time: 50
  period: 30
  deadline: 100
  priority: 2

- id: 2
  worst-case execution time: 10
  arrival curve: [220,[[1,1],[105,2]]]
  deadline: 100
  priority: 1
```

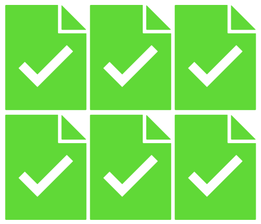
POET supports arbitrary arrival curves

POET: FOUNDATIONAL RTA BASED ON PROSA

Generate proof-carrying response-time bounds based on Prosa and Coq



CERTIFICATES



≃

Coq Certificates

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

CERTIFICATES

$\simeq$

Coq Certificates

Task set

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

CERTIFICATES

≃

Coq Certificates

Task set

Task under analysis

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].

Let tsk := tsk1.
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

CERTIFICATES

≃

Coq Certificates

Task set

Task under analysis

Response time

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].

Let tsk := tsk1.

...
Let R := 10%N.
...
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

CERTIFICATES

≅

Coq Certificates

Task set

Task under analysis

Response time

Fix-point equation

Auto-generated proof

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].

Let tsk := tsk1.

...
Let R := 10%N.
...
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

```
Lemma R_is_maximum:
  ∀ A, is_in_search_space tsk L A →
    ∃ F, rbf tsk (A + ε) + rbf ts tsk (A + F) ≤ A + F
         ∧ F ≤ R.
Proof.
  move ⇒ A SS.
  ...
```

CERTIFICATES

≅

Coq Certificates

Task set

Task under analysis

Response time

Fix-point equation

Auto-generated proof

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].

Let tsk := tsk1.

...
Let R := 10%N.
...

Lemma R_is_maximum:
  ∀ A, is_in_search_space tsk L A →
    ∃ F, rbf tsk (A + ε) + rbf ts tsk (A + F) ≤ A + F
         ∧ F ≤ R.
Proof.
  move ⇒ A SS.
  ...
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

Prosa: this equation implies the RTA theorem

POET: given task set satisfies this equation

CERTIFICATES

≅

Coq Certificates

Task set

```
Let tsk1 := {|
  task_id := 1;
  task_cost := 10;
  task_period := Period 30;
  task_deadline := 100;
  task_priority := 2 |}.
Let tsk2 := ...
Let ts := [::tsk01,tsk02].
```

Task under analysis

```
Let tsk := tsk1.
```

Response time

```
...
Let R := 10%N.
...
```

```
- id: 1
  worst-case execution time: 10
  period: 30
  deadline: 100
  priority: 2
```

Fix-point equation

```
Lemma R_is_maximum:
  ∀ A, is_in_search_space tsk L A →
    ∃ F, rbf tsk (A + ε) + rbf ts tsk (A + F) ≤ A + F
         ∧ F ≤ R.
```

Auto-generated proof

```
Proof.
  move ⇒ A SS.
  ...
```

Final conclusion

```
Theorem R_respects_deadlines :
   task_response_time_bound arr_seq sched tsk R
   ∧ R ≤ task_deadline tsk.
```

Auto-generated proof, again

```
Proof.
  ...
```

CERTIFICATES ARE EXPLAINABLE EVIDENCE OF TIMELINESS

Coq Certificates

CERTIFICATES ARE EXPLAINABLE EVIDENCE OF TIMELINESS

Coq Certificates

- Certificates are **short** and **readable** Coq files
- Certificates are generated **automatically**
  - → Users obtain formally-verified results without having to know Coq

CERTIFICATES ARE EXPLAINABLE EVIDENCE OF TIMELINESS

Coq Certificates

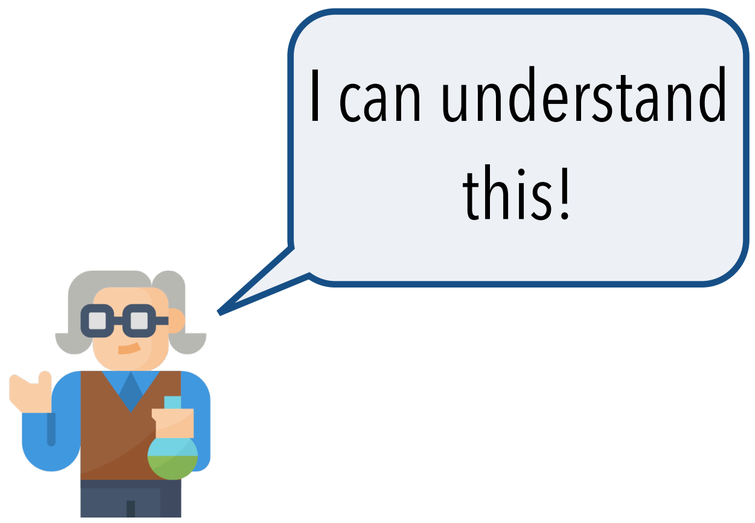
- Certificates are **short** and **readable** Coq files
- Certificates are generated **automatically**
  - → Users obtain formally-verified results without having to know Coq
- Certificates can be **studied** and **dissected** up until their fundamental axioms
  - → POET generates **explainable** evidence of timeliness

CERTIFICATES ARE EXPLAINABLE EVIDENCE OF TIMELINESS

Coq Certificates

- Certificates are **short** and **readable** Coq files
- Certificates are generated **automatically**
  - → Users obtain formally-verified results without having to know Coq
- Certificates can be **studied** and **dissected** up until their fundamental axioms
  - → POET generates **explainable** evidence of timeliness

```
Theorem R_respects_deadlines :
   task_response_time_bound arr_seq sched tsk R
   ∧ R ≤ task_deadline tsk.
Proof.
  ...
```



CERTIFICATES ARE EXPLAINABLE EVIDENCE OF TIMELINESS

Coq Certificates

- Certificates are **short** and **readable** Coq files
- Certificates are generated **automatically**
  - → Users obtain formally-verified results without having to know Coq

- Certificates can be **studied** and **dissected** up until their fundamental axioms
  - → POET generates **explainable** evidence of timeliness

- Certificates **do not depend** on POET
  - → We do not need to verify or trust POET

CURRENT CAPABILITIES OF POET

POET currently supports:

- Scheduling policies: *Earliest-deadline first (EDF)*, *Fixed-priority (FP)*
- Preemption policies: *Fully preemptive*, *Fully non-preemptive*
- Workload: *Periodic* and *sporadic* with arbitrary *arrival curves*
- Tasks with *arbitrary deadlines*

…see paper for details!

EVALUATION

EVALUATION: SETUP

Goal: assess scalability of the proposed approach w.r.t. such as number of tasks and utilization

EVALUATION: SETUP

Goal: assess scalability of the proposed approach w.r.t. such as number of tasks and utilization

- We evaluated **performance of Coq** on ≈ 7000 randomly-generated task sets

EVALUATION: SETUP

Goal: assess scalability of the proposed approach w.r.t. such as number of tasks and utilization

- We evaluated **performance of Coq** on ≈ 7000 randomly-generated task sets
- **Scheduling** / **preemption policy**: {FP, EDF} $\times$ {Fully-Pr, Fully Non-Pr}
- **Cardinality**: 2-40 with step 2 (50 tasks in the paper)
- **Utilization**: 50%, 60%, 70%, 80%, 90%

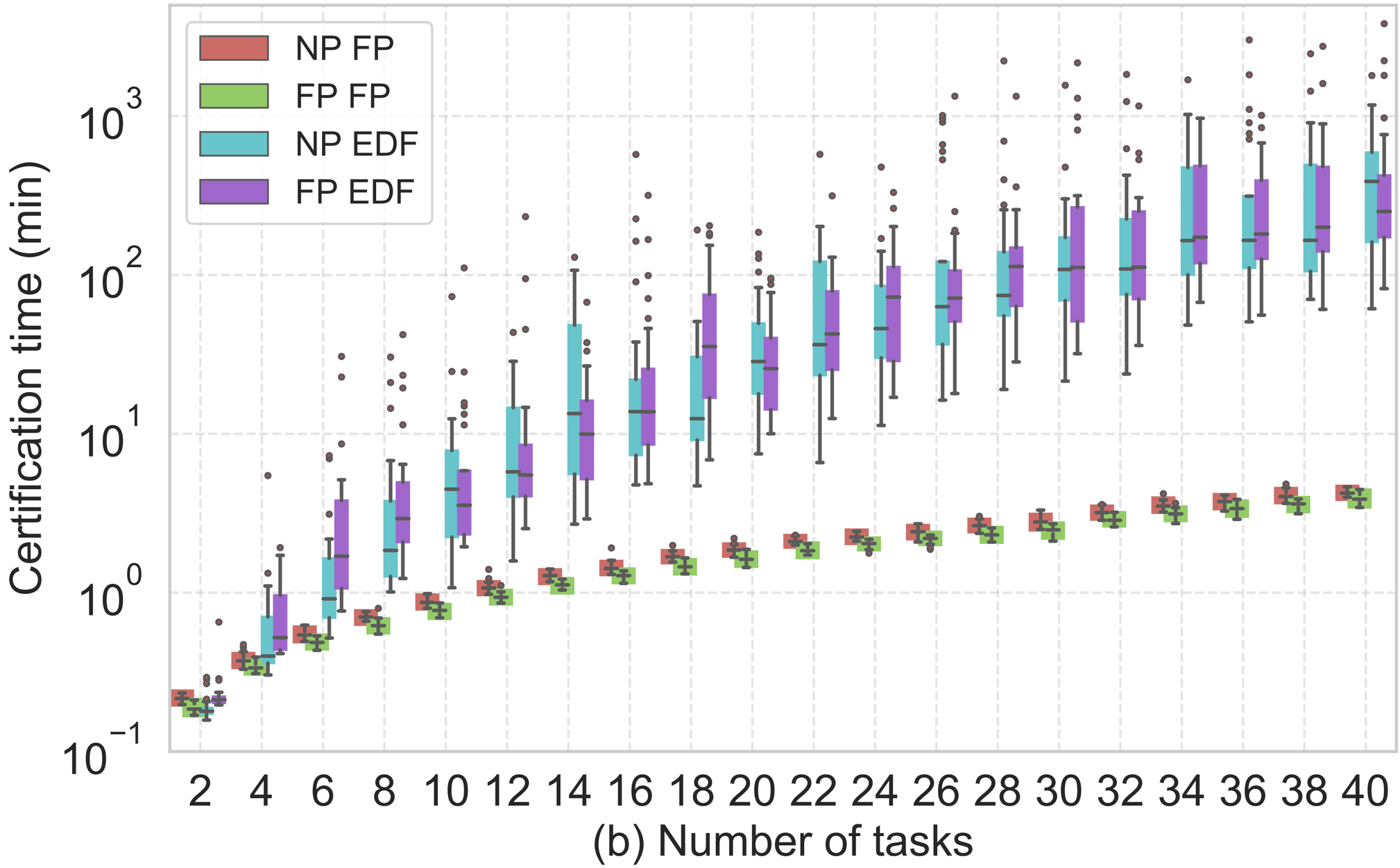
EVALUATION: SETUP

Goal: assess scalability of the proposed approach w.r.t. such as number of tasks and utilization

- We evaluated **performance of Coq** on ≈ 7000 randomly-generated task sets
- **Scheduling** / **preemption policy**: {FP, EDF} $\times$ {Fully-Pr, Fully Non-Pr}
- **Cardinality**: 2-40 with step 2
- **Utilization**: 50%, 60%, 70%, 80%, 90%
- **Workload**:
  - **This talk**: mixed workload
  - **Paper**: two move workload-types

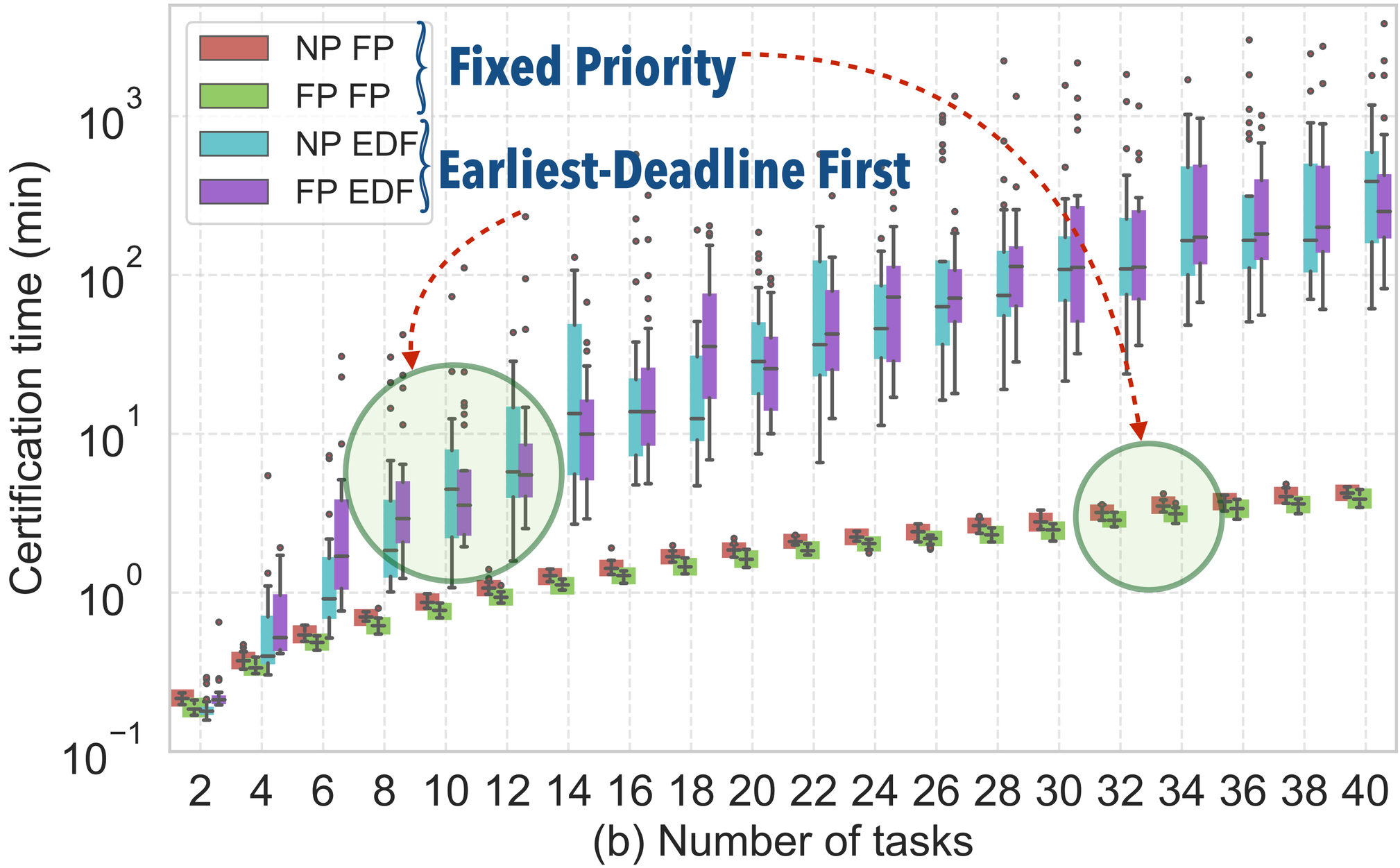
(defined by arrival curves)

BRIEF OVERVIEW



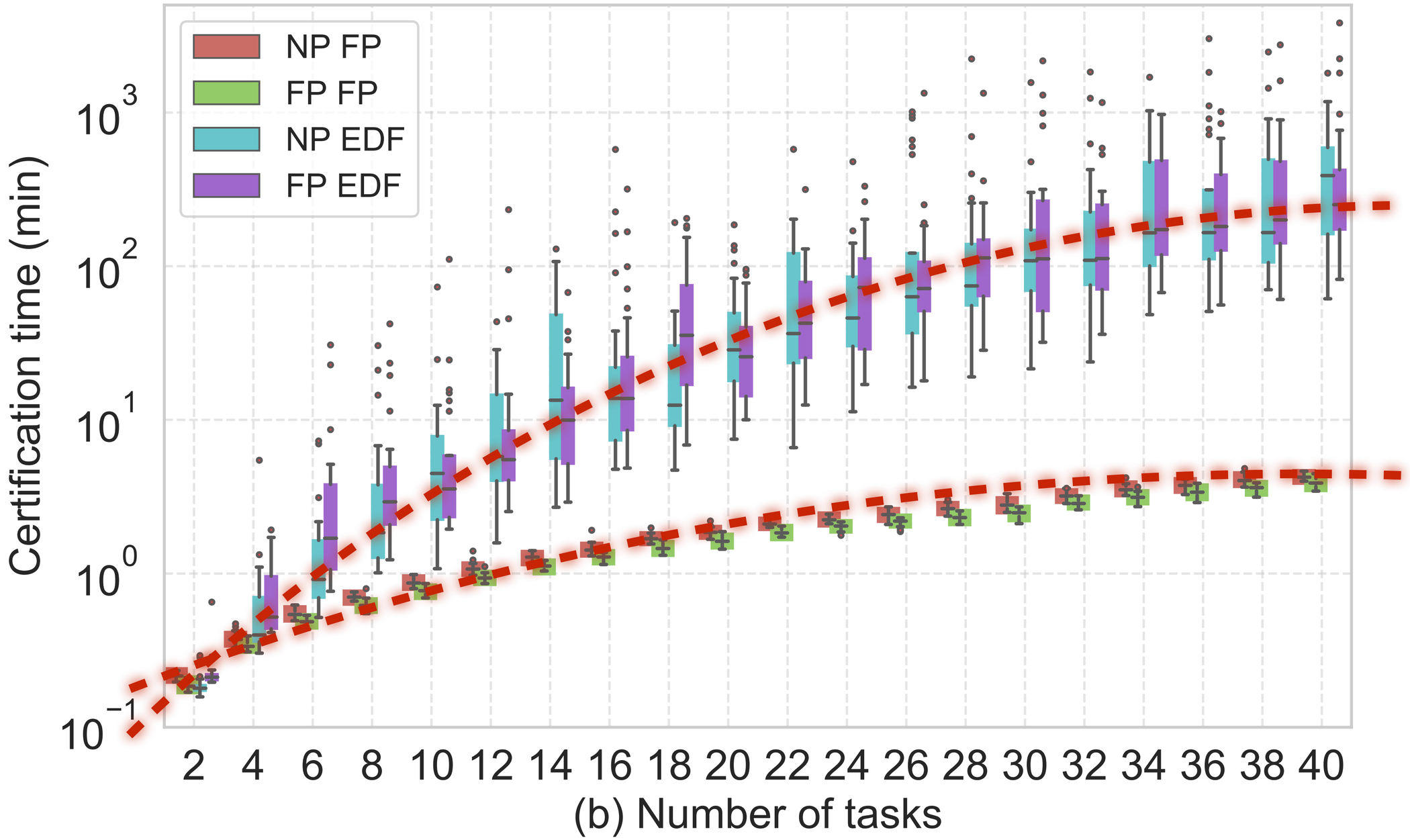
Workload: Mixed workload **# Task sets**: 2000 task sets

BRIEF OVERVIEW



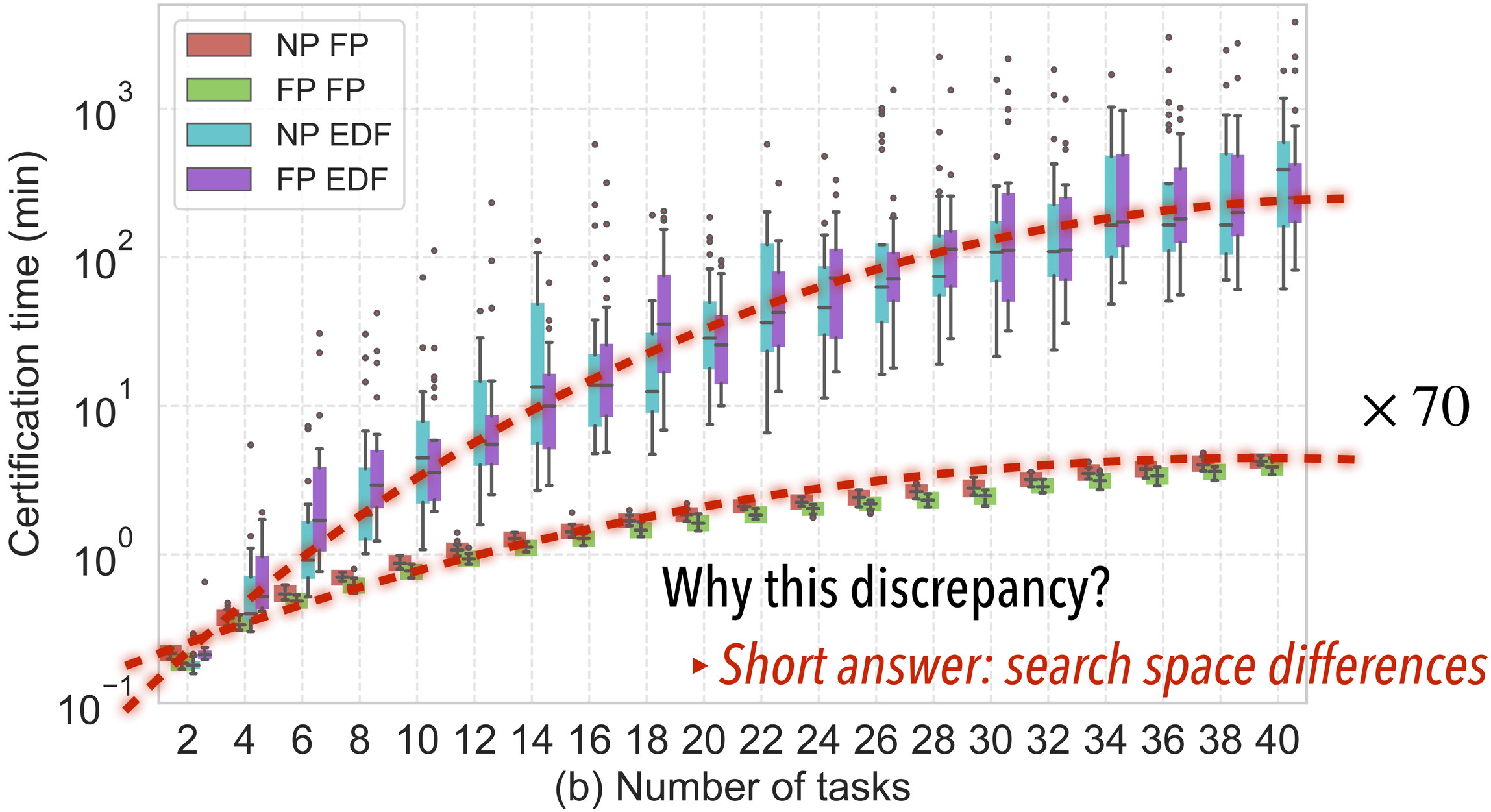
Workload: Mixed workload **# Task sets**: 2000 task sets

OVERALL TREND



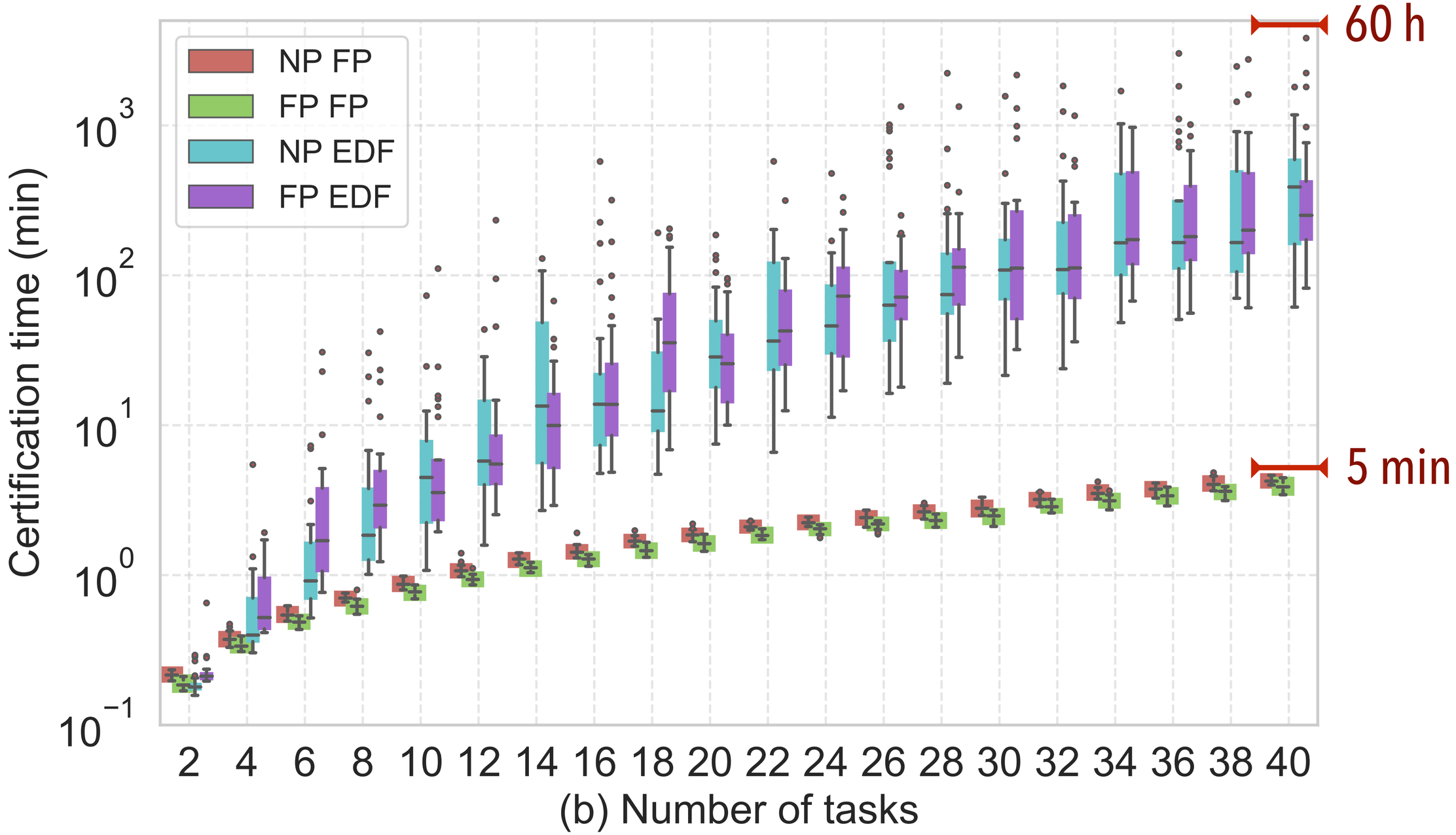
Workload: Mixed workload **# Task sets**: 2000 task sets

OVERALL TREND



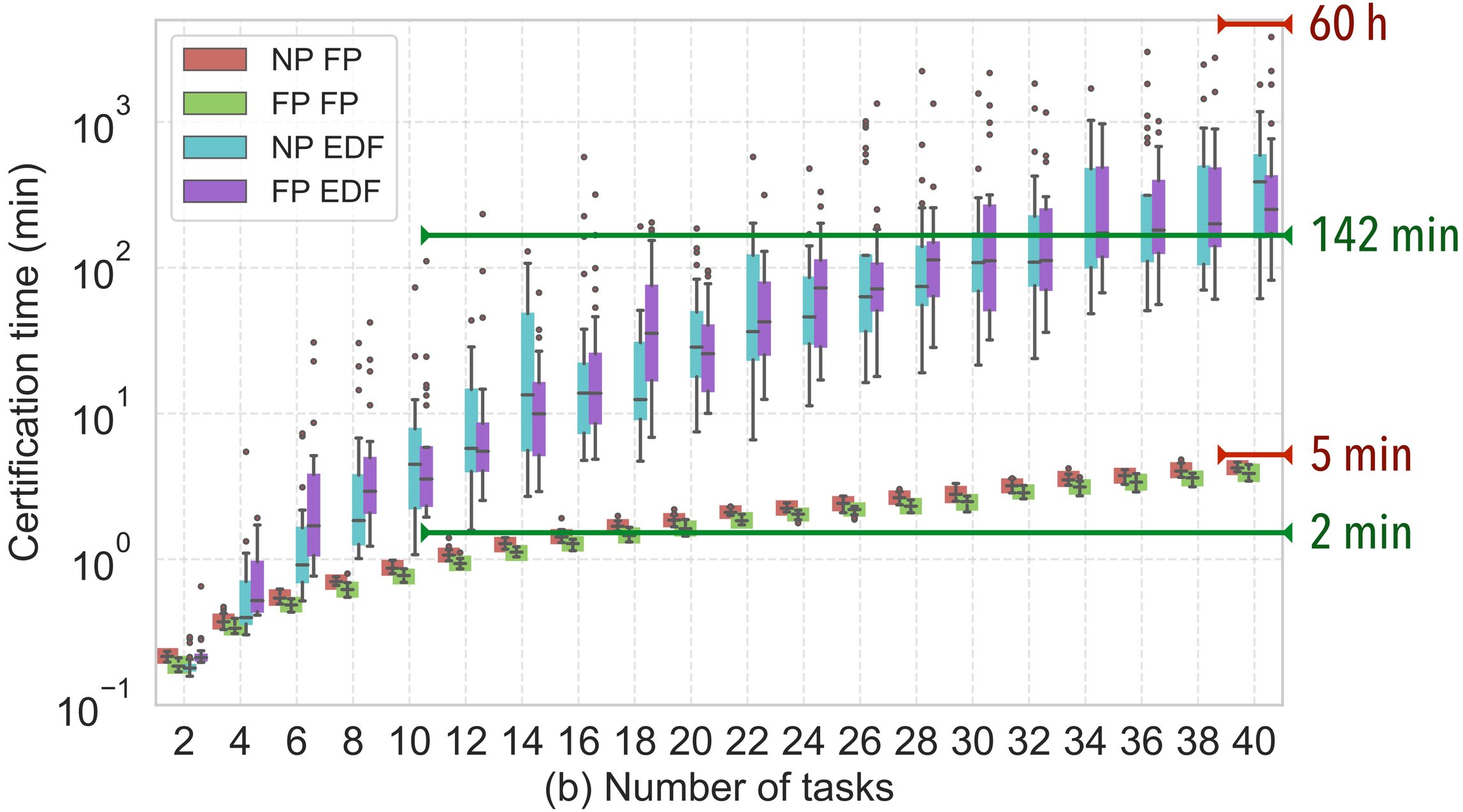
Workload: Mixed workload **# Task sets**: 2000 task sets

MAXIMUM RUNTIME



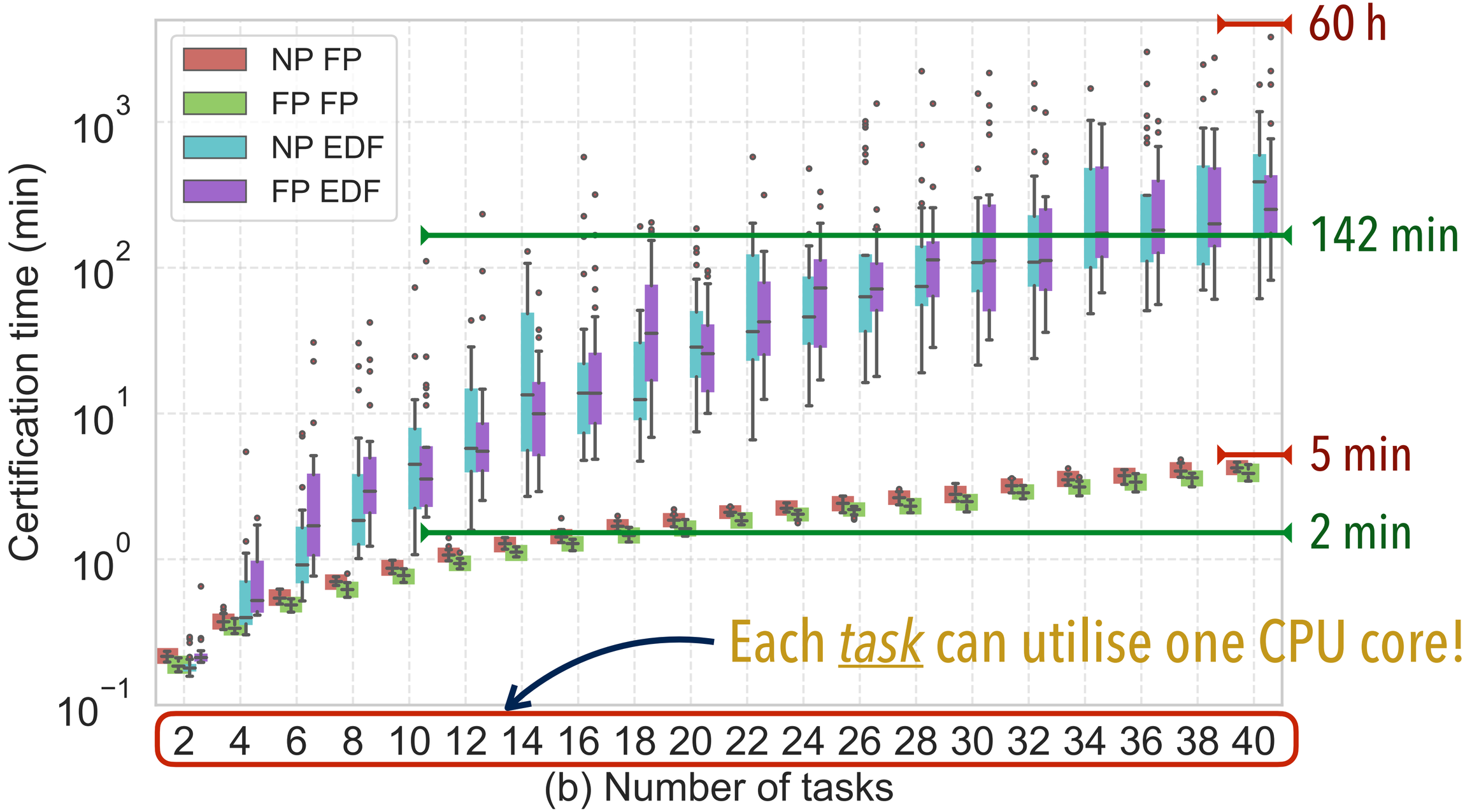
Workload: Mixed workload **# Task sets**: 2000 task sets

AVERAGE RUNTIME



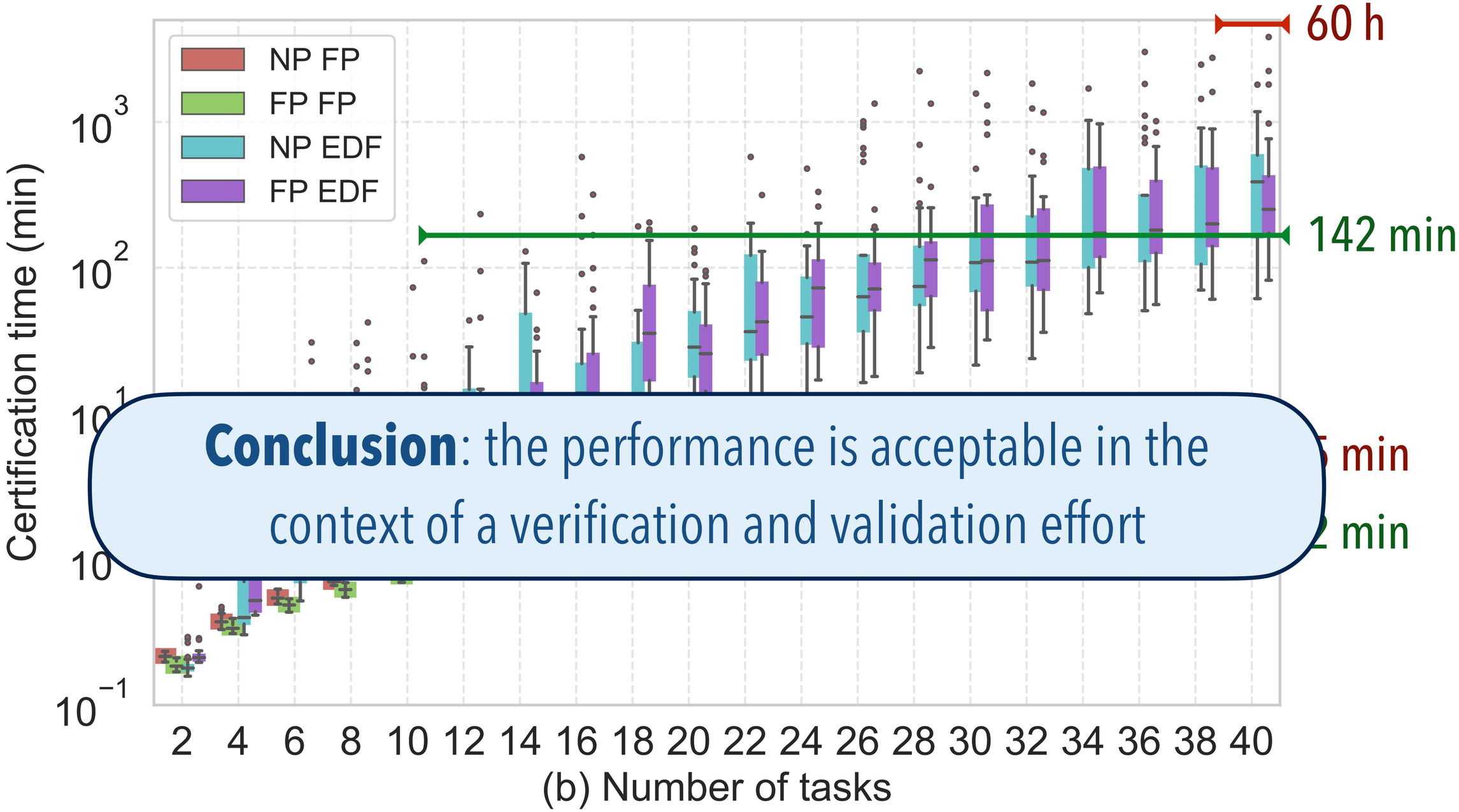
Workload: Mixed workload **# Task sets**: 2000 task sets

VERIFICATION IS EASILY PARALLELIZABLE



Workload: Mixed workload **# Task sets**: 2000 task sets

OVERALL CONCLUSION



Conclusion: the performance is acceptable in the context of a verification and validation effort

Workload: Mixed workload **# Task sets**: 2000 task sets

CONCLUSION

SEE PAPER FOR IMPORTANT DETAILS

"Naive" proof-oriented computation in Coq is **very slow**

- *How to speed it up?*

SEE PAPER FOR IMPORTANT DETAILS

"Naive" proof-oriented computation in Coq is **very slow**

- *How to speed it up?*

Coq cannot automatically find potential **contradictions** in analysis assumptions

- *How to defend against conflicting hypotheses?*

SEE PAPER FOR IMPORTANT DETAILS

"Naive" proof-oriented computation in Coq is **very slow**

- *How to speed it up?*

Coq may accept **truncated certificates**

- *How to engineer POET in light of this?*

Coq cannot automatically find potential **contradictions** in analysis assumptions

- *How to defend against conflicting hypotheses?*

SEE PAPER FOR IMPORTANT DETAILS

"Naive" proof-oriented computation in Coq is **very slow**

- *How to speed it up?*

Coq may accept **truncated certificates**

- *How to engineer POET in light of this?*

Coq cannot automatically find potential **contradictions** in analysis assumptions

- *How to defend against conflicting hypotheses?*

Support for **arbitrary arrival curves** was not trivial

- *How to compute the search space efficiently?*

POET, THE FIRST FOUNDATIONAL RTA TOOL

Summary:

- POET produces **formally-verified** response-time bounds
- Bounds are proven correct by **automatically generated Coq proofs**
- Users **do not need to know** formal verification

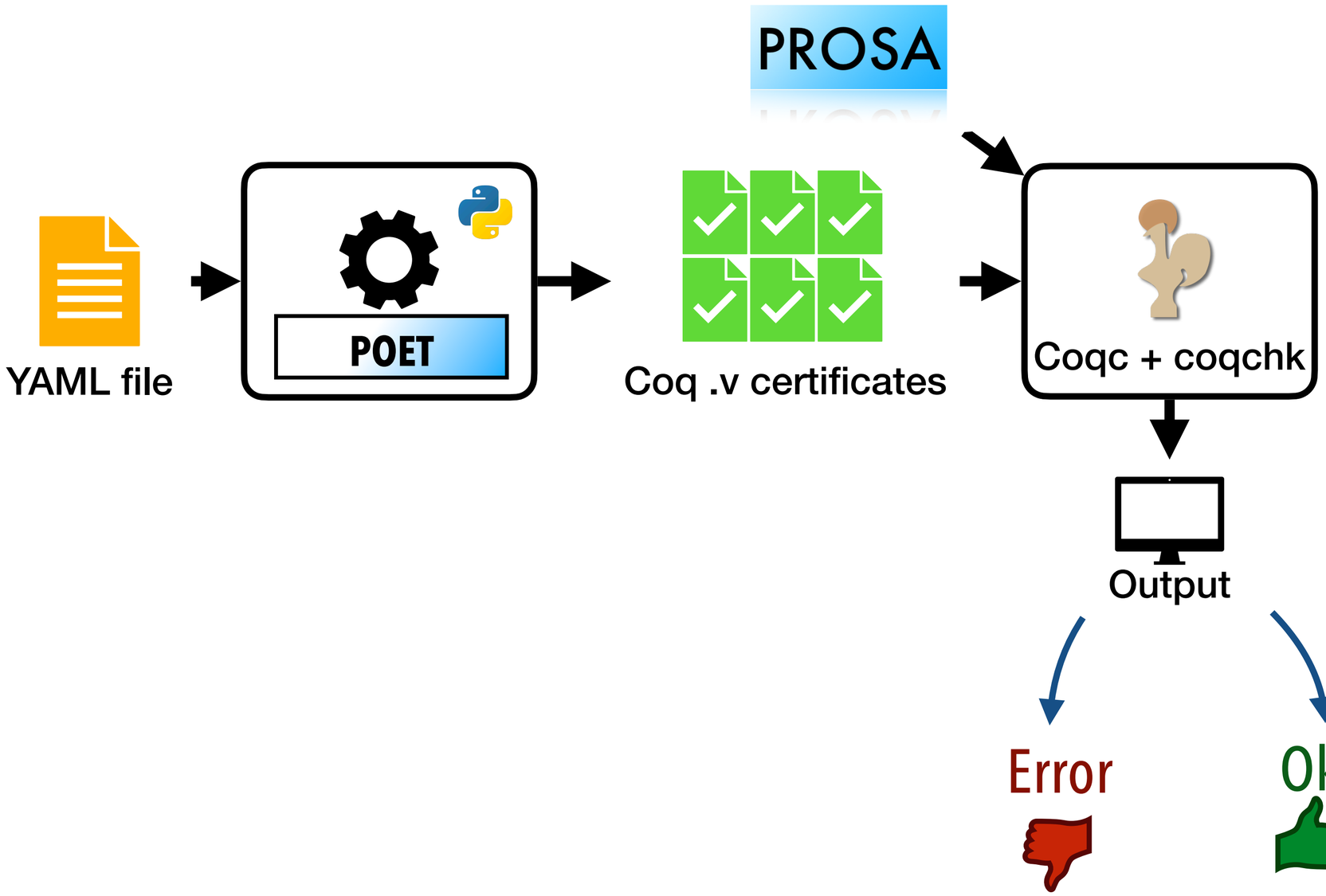
POET, THE FIRST FOUNDATIONAL RTA TOOL

Summary:

- POET produces **formally-verified** response-time bounds
- Bounds are proven correct by **automatically generated Coq proofs**
- Users **do not need to know** formal verification

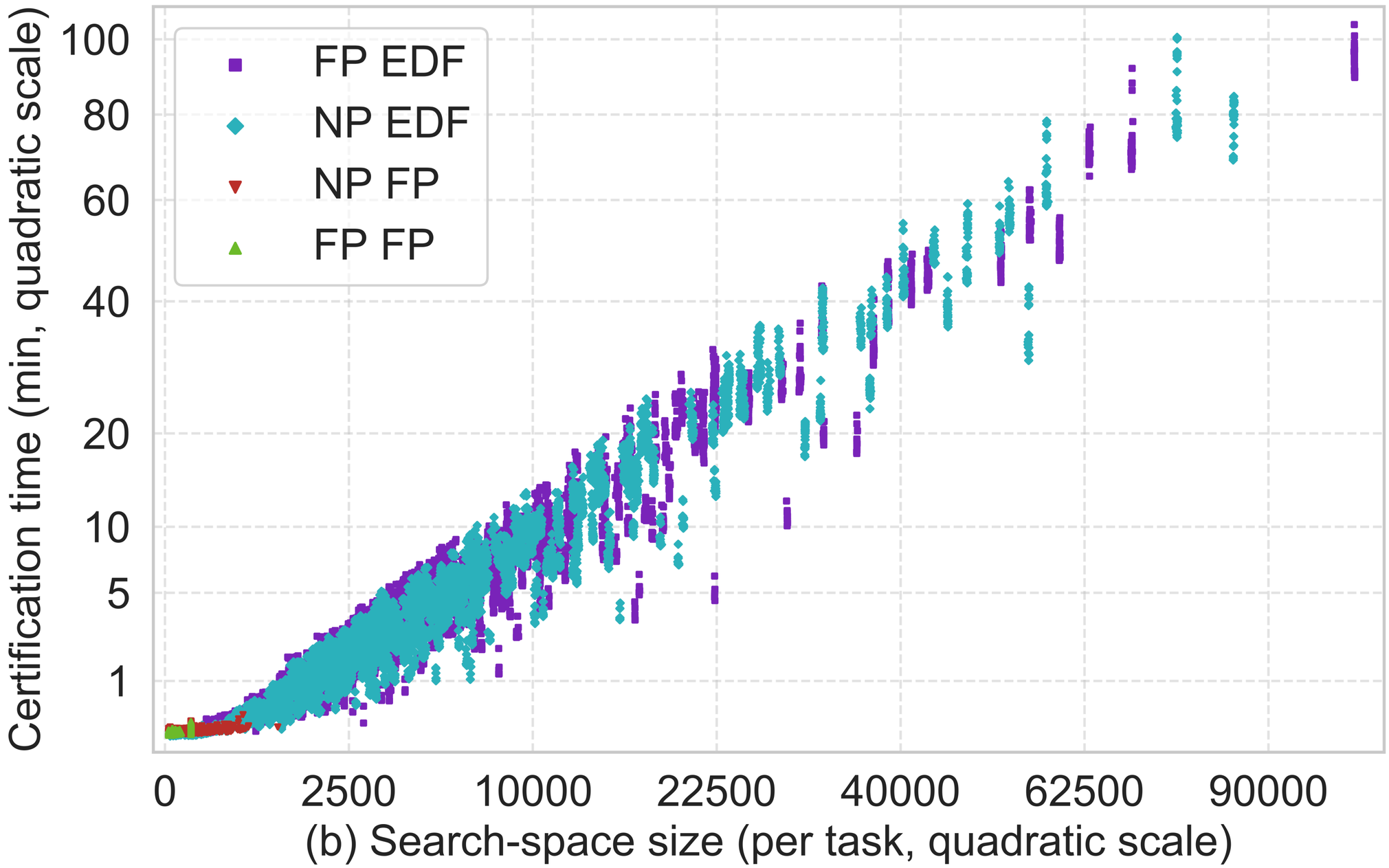
Future work:

- More **complex workloads**: synchronization and precedence constraints
- **Realistic system models**: scheduling overheads and multiprocessor platforms


PROSA
YAML file
POET
Coq .v certificates
Coqc + coqchk
Output
Error
Ok

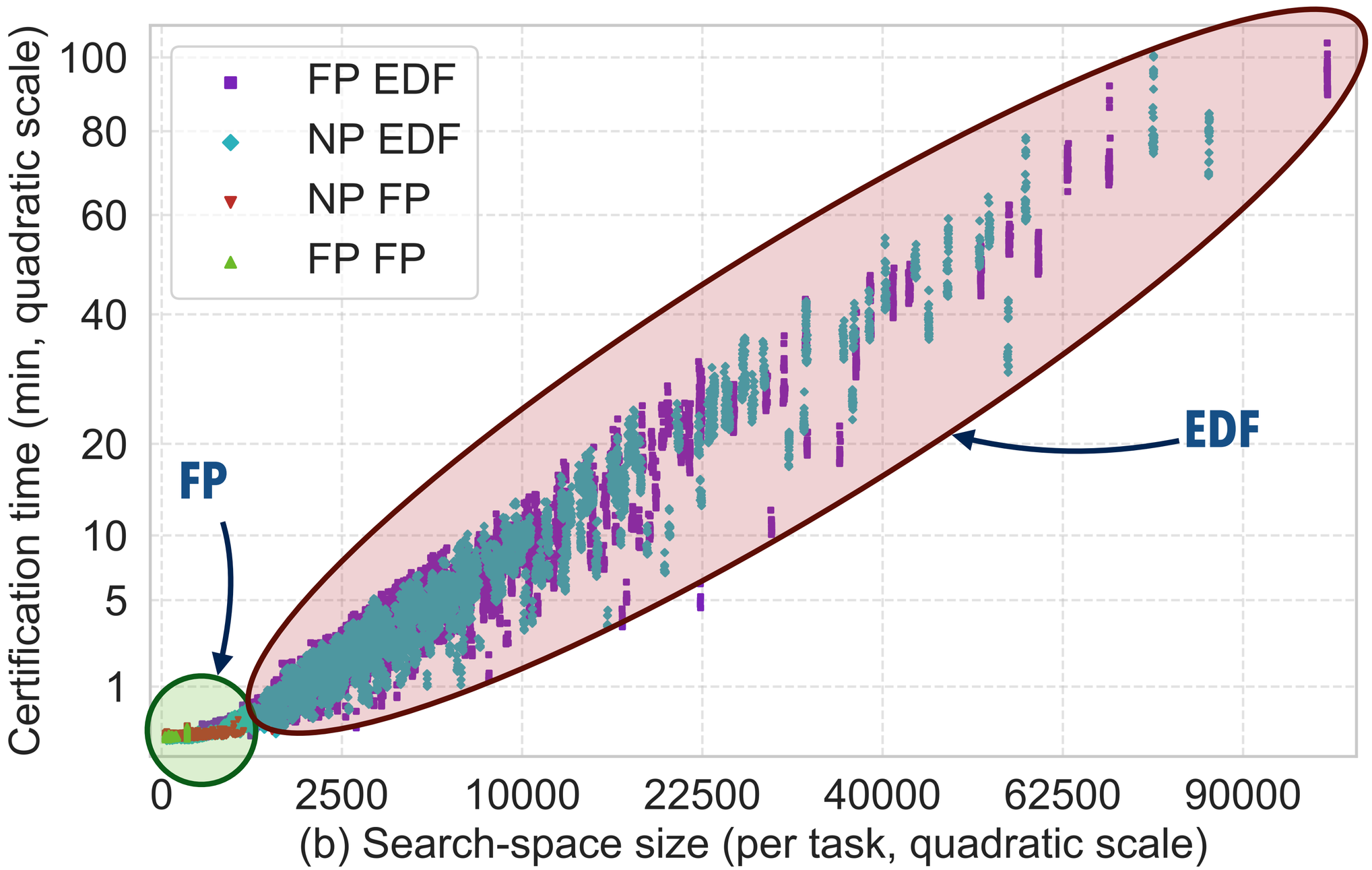
Back up slides

DIFFERENCE IN SEARCH SPACE SIZE



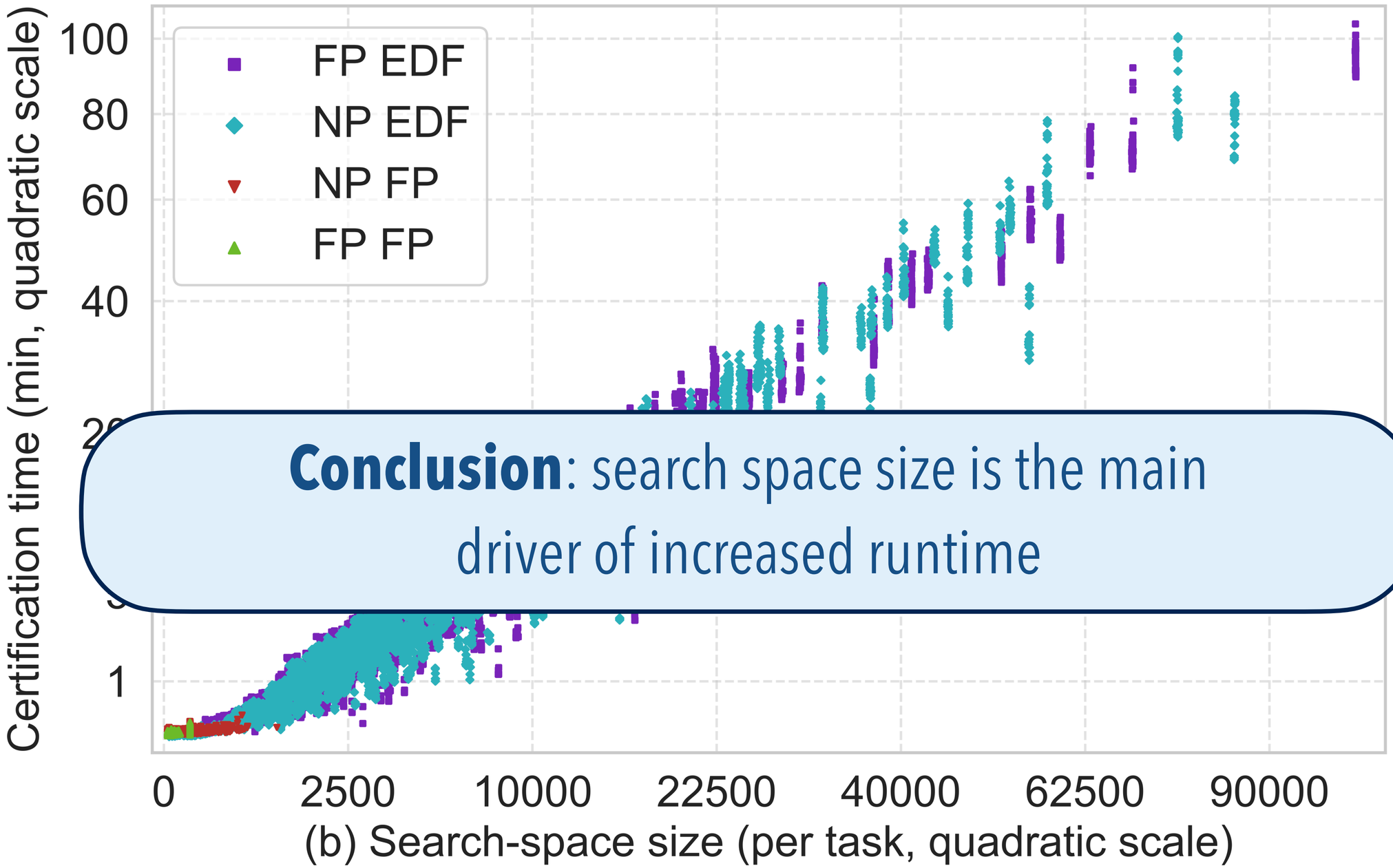
Workload: Mixed workload **# Task sets**: 2000 task sets

DIFFERENCE IN SEARCH SPACE SIZE



Workload: Mixed workload **# Task sets**: 2000 task sets

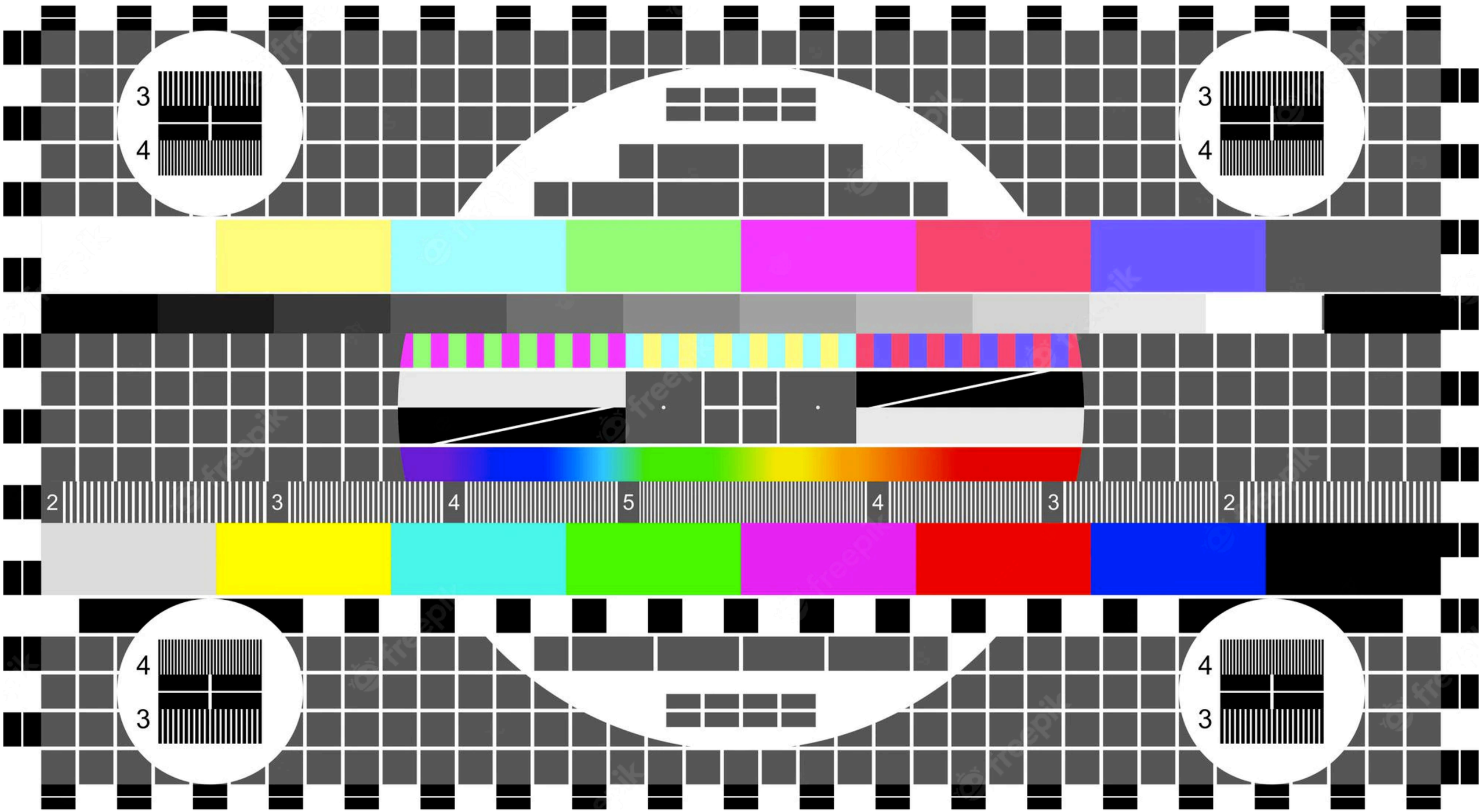
DIFFERENCE IN SEARCH SPACE SIZE



Conclusion: search space size is the main driver of increased runtime

Workload: Mixed workload

 # Task sets: 2000 task sets


3
4
3
4
2
3
4
5
4
3
2
4
3
4
3