Prosa: A Case for Readable Mechanized Schedulability Analysis

Felipe Cerqueira, Felix Stutz, Björn B. Brandenburg

Max Planck Institute for Software Systems

ECRTS 2016
5-8 July
Toulouse, France

Prosa

Open-source foundation for formally **pro**ven **s**chedulability **a**nalysis

Jobs

Multiprocessor

Response time

…

Readable Formal Specification

Schedulability Tests

Response-time Bounds

…

Mechanized Proofs

This Talk

This Talk

Mechanized proofs provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

This Talk

Mechanized proofs provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

By focusing on **readability** and by **maintaining the established proof culture,** mechanized proofs **can reach the community at large.**

This Talk

Mechanized proofs provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

By focusing on **readability** and by **maintaining the established proof culture,** mechanized proofs **can reach the community at large.**

Thanks to **mature proof assistants and libraries**, Prosa allows mechanizing **recent and complex** schedulability analyses in **reasonable time.**

Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa

Outline of the Talk

Why mechanized proofs?

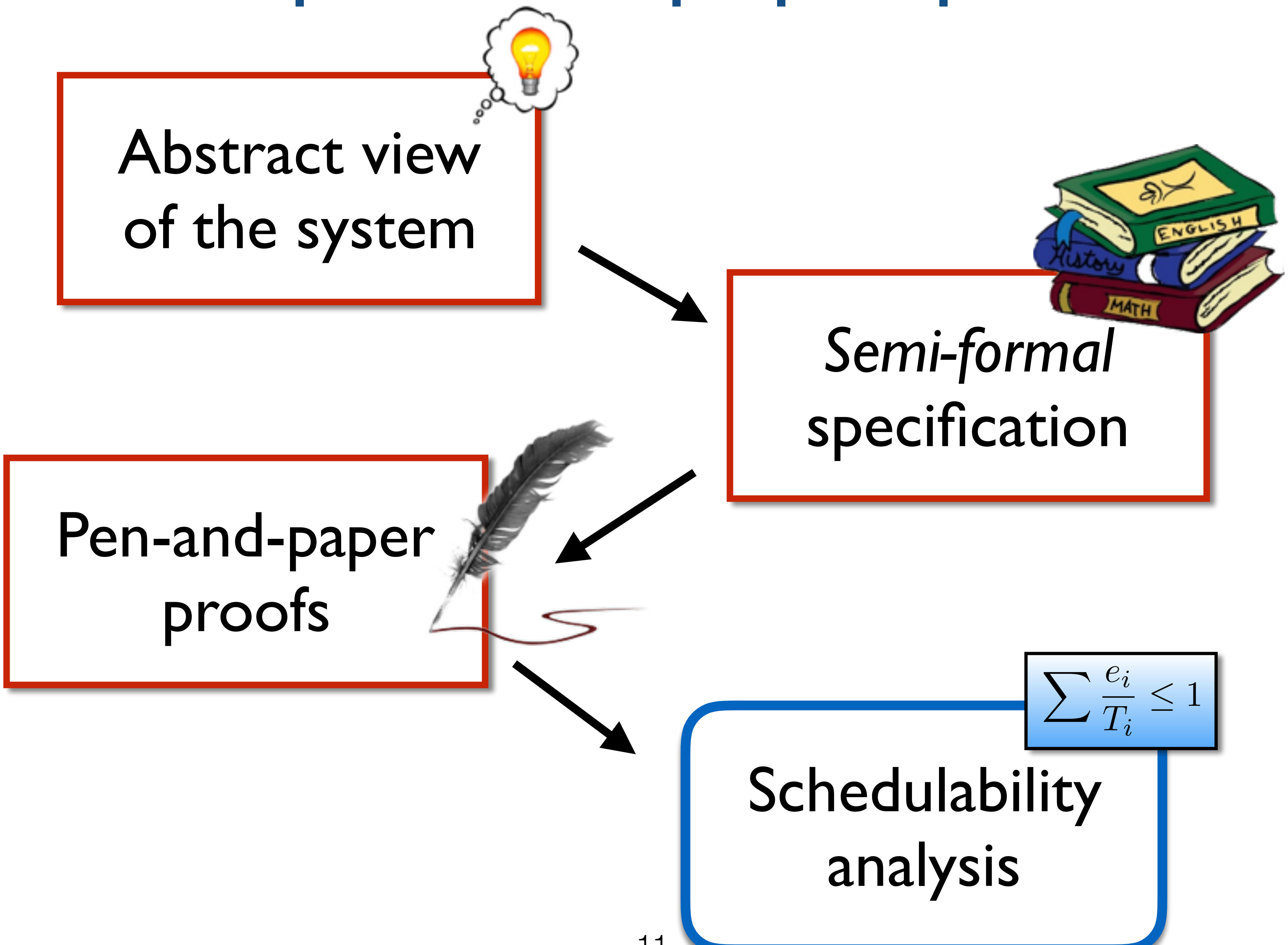
Challenges & Principles

A Taste of Prosa

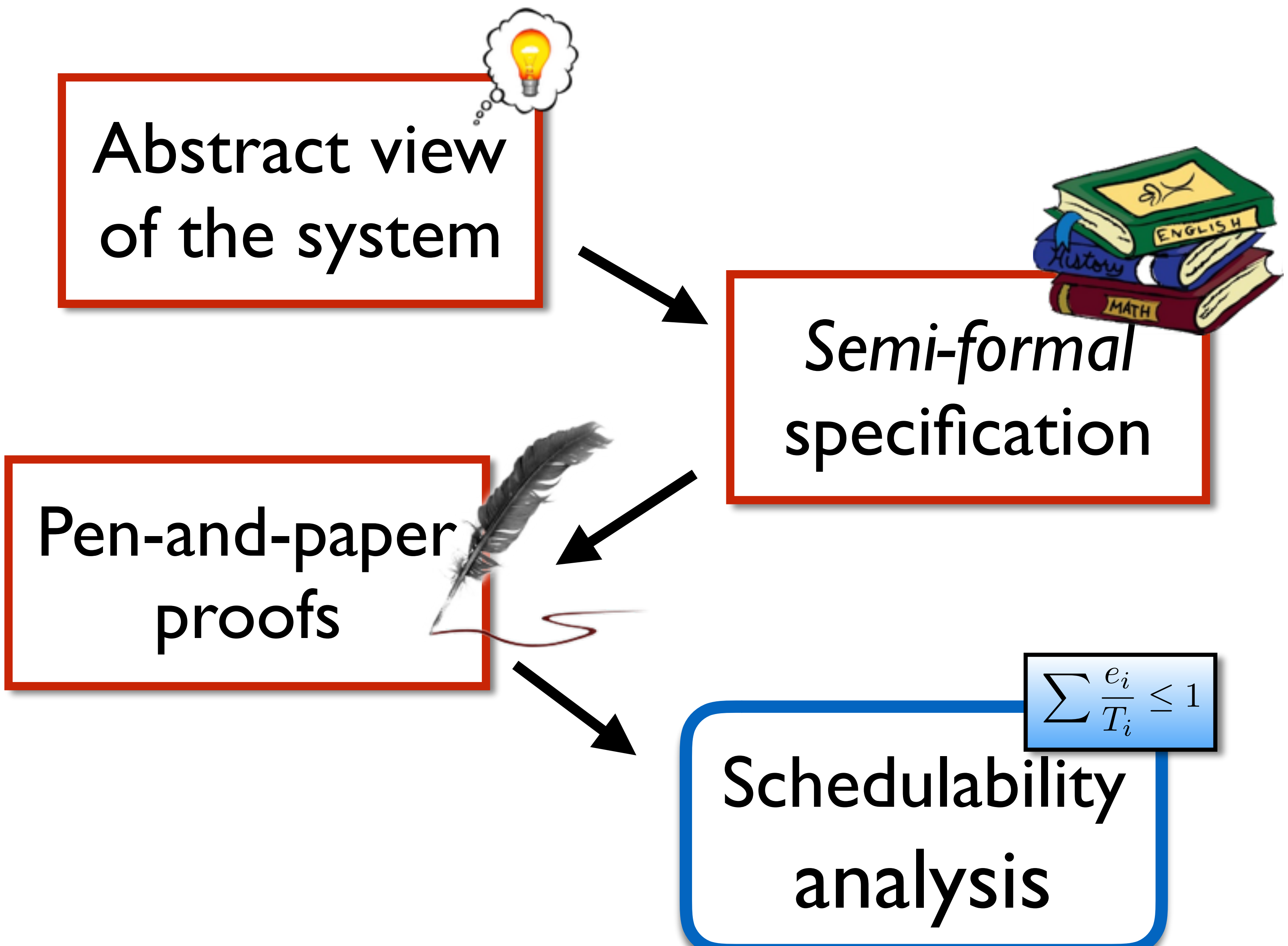
What do we mean by **mechanized?**

RTS theory has been built with pen-and-paper proofs

RTS theory has been built with pen-and-paper proofs

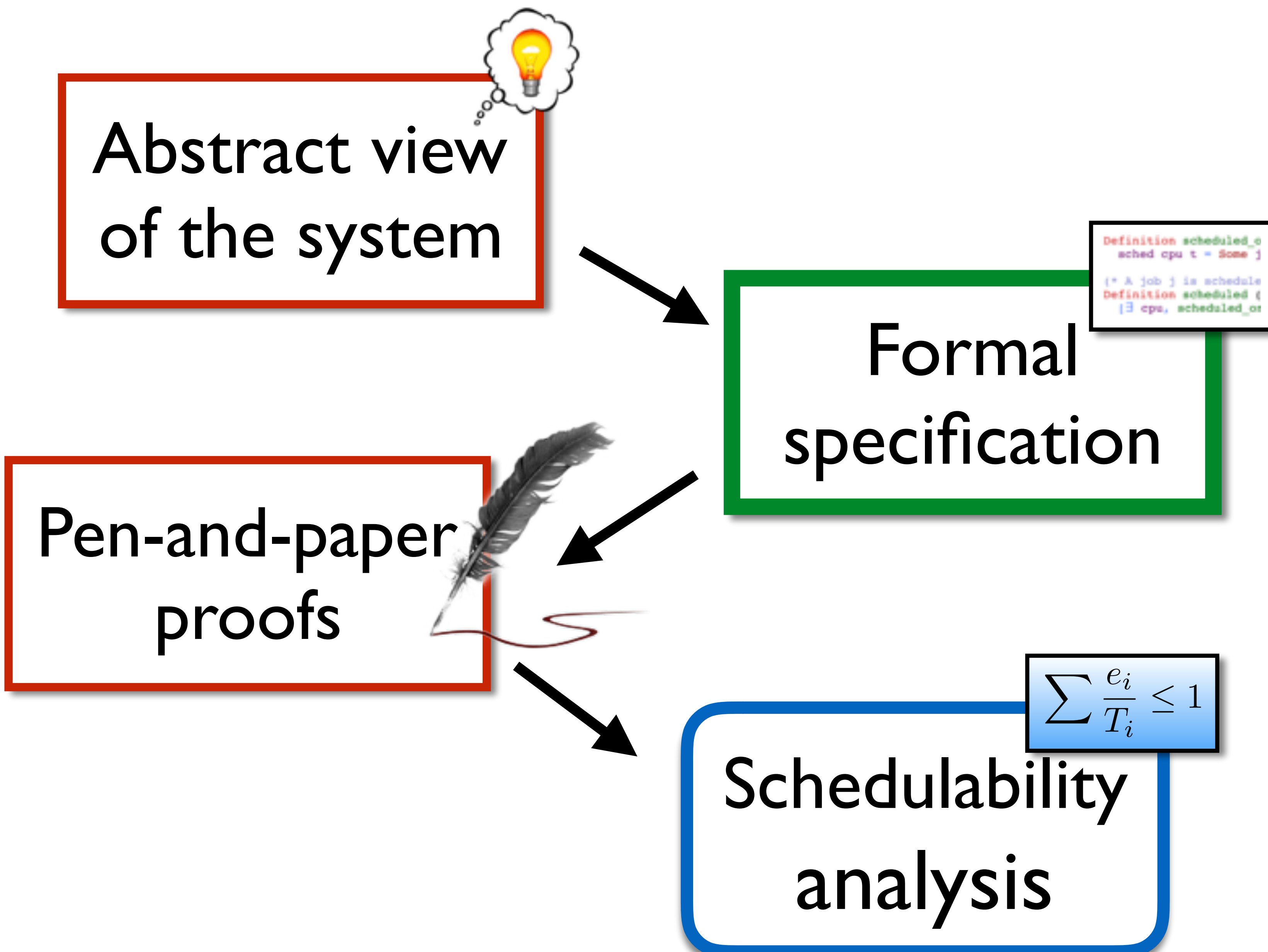


What is **mechanized** schedulability analysis?



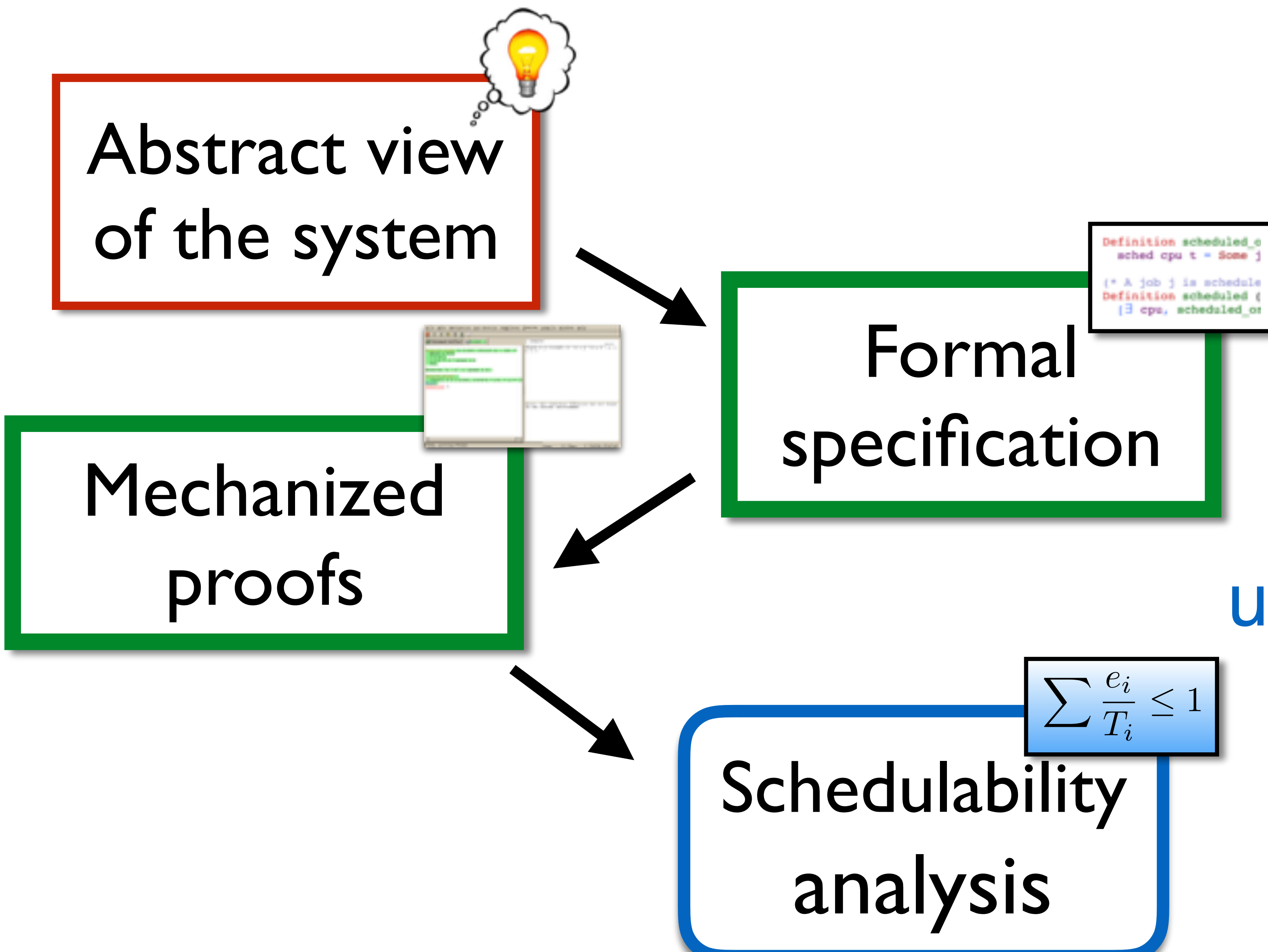
What is the difference?

What is **mechanized** schedulability analysis?



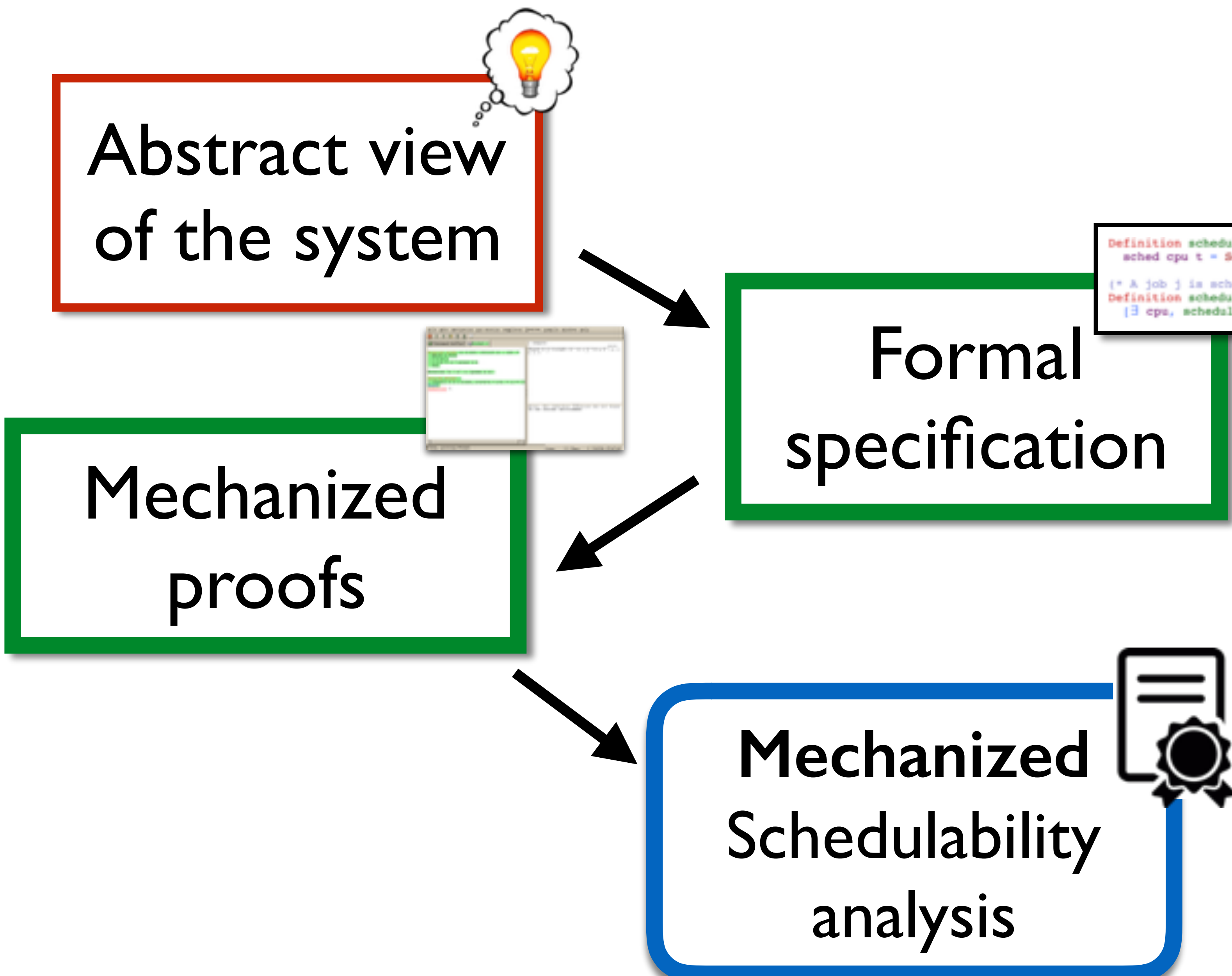
We switch to a **formal specification**

What is **mechanized** schedulability analysis?



We prove theorems using a **proof assistant**

What is **mechanized** schedulability analysis?



The resulting schedulability analysis is **formally verified**

Why mechanized proofs?

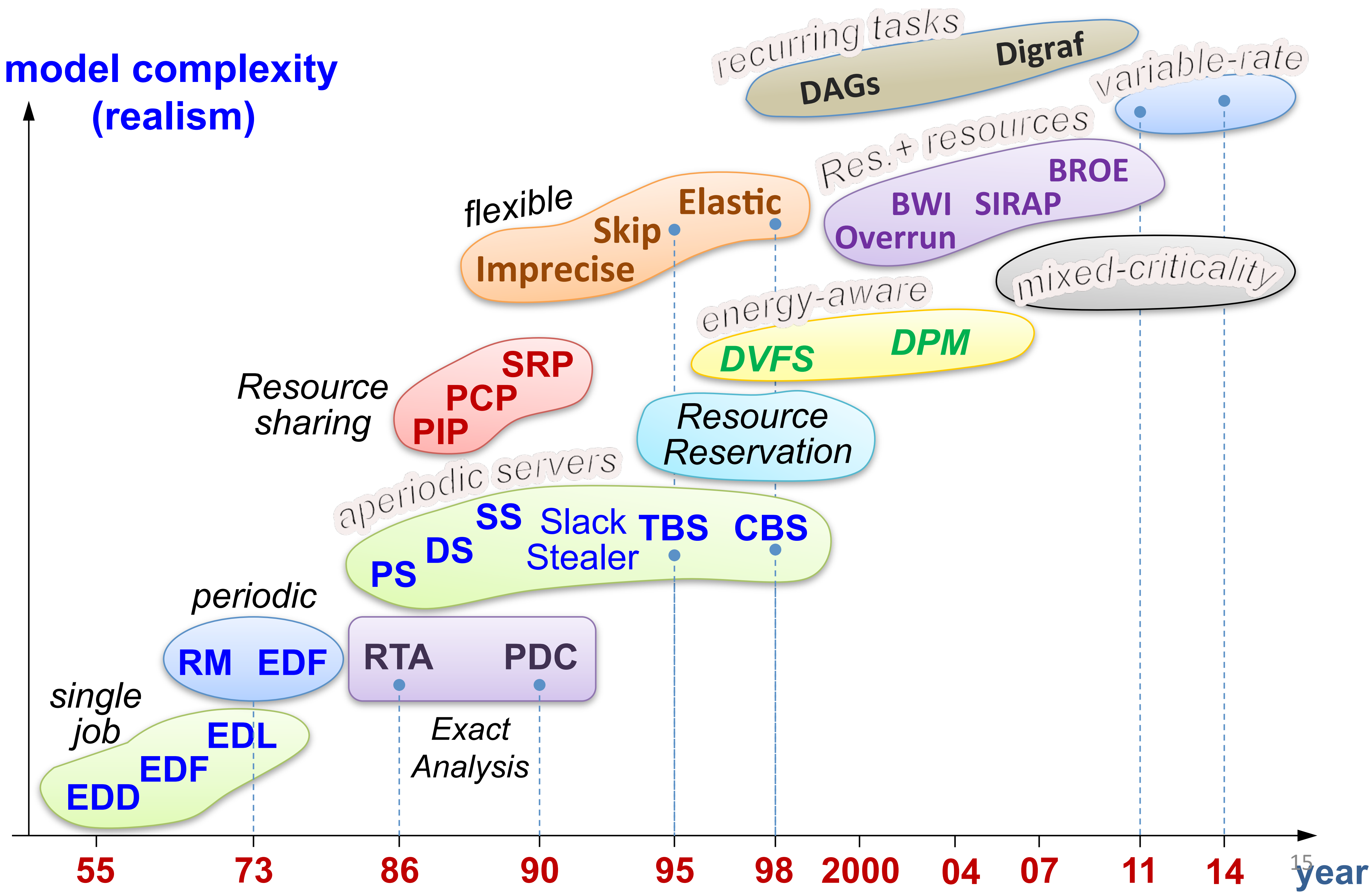
Why mechanized proofs?

Guaranteed Correctness

Trustworthy Extensions

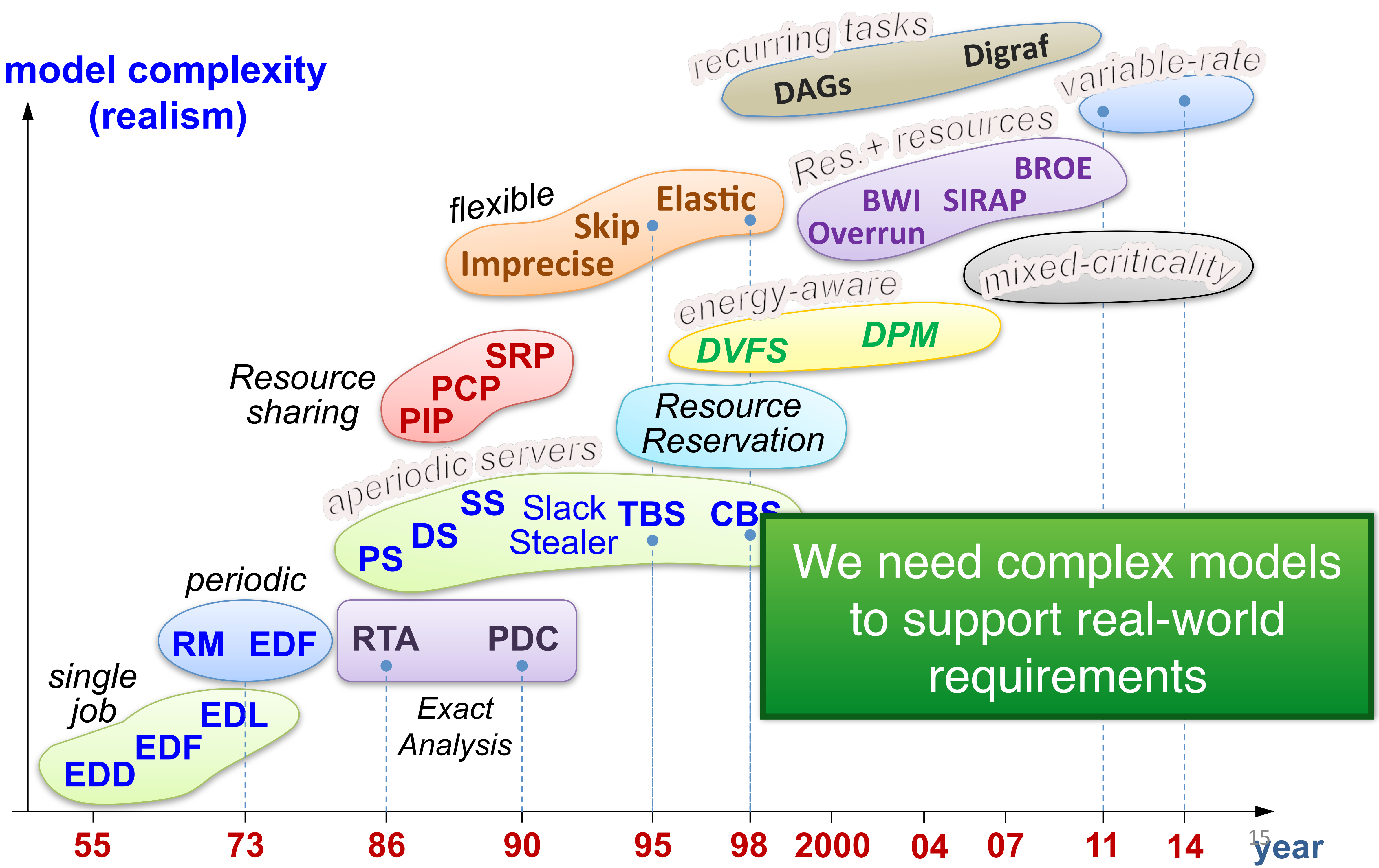
Safe Composition

RTS have become more complex



Source: G. Buttazzo (Keynote @ RTSS'14)

RTS have become more complex



Source: G. Buttazzo (Keynote @ RTSS'14)

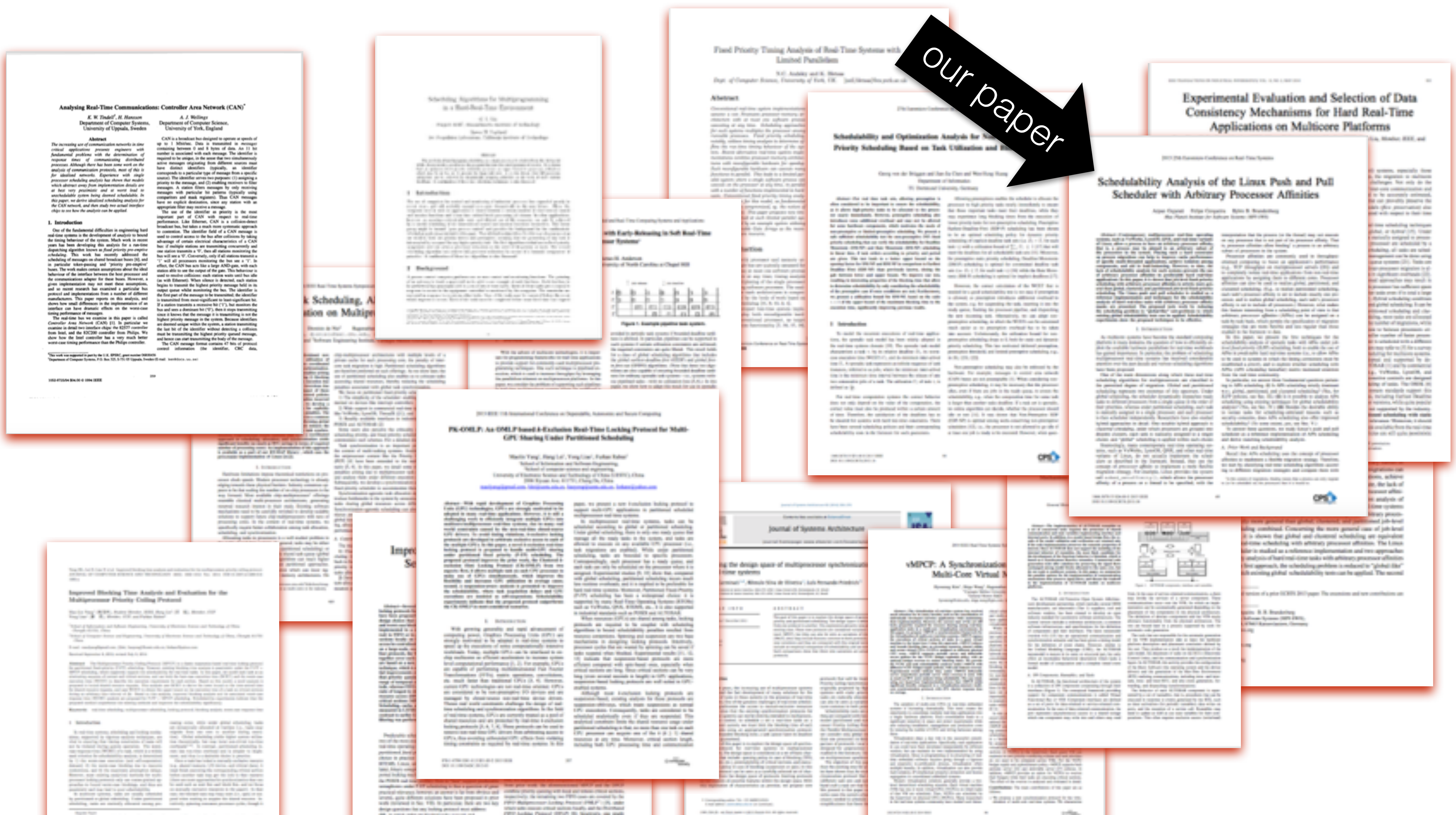
This complexity comes with a price

The original analysis for CAN **had a bug** that remained undetected from 1994 to 2006 [1].

[1] Davis, R. I., Burns, A., Bril, R. J., & Lukkien, J. J. “Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.” Real-Time Systems, 35(3), 239-272, 2007.

Bugs are no longer an exception

Proofs have become so complicated
that they often contain bugs.



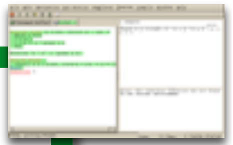
Analysis for safety-critical systems?

How to ensure that schedulability analysis is actually correct?

Analysis for safety-critical systems?

How to ensure that schedulability analysis is actually correct?

Mechanized proofs



Opportunity: correctness is **inherently guaranteed.**

Why mechanized proofs?

Guaranteed Correctness

Trustworthy Extensions

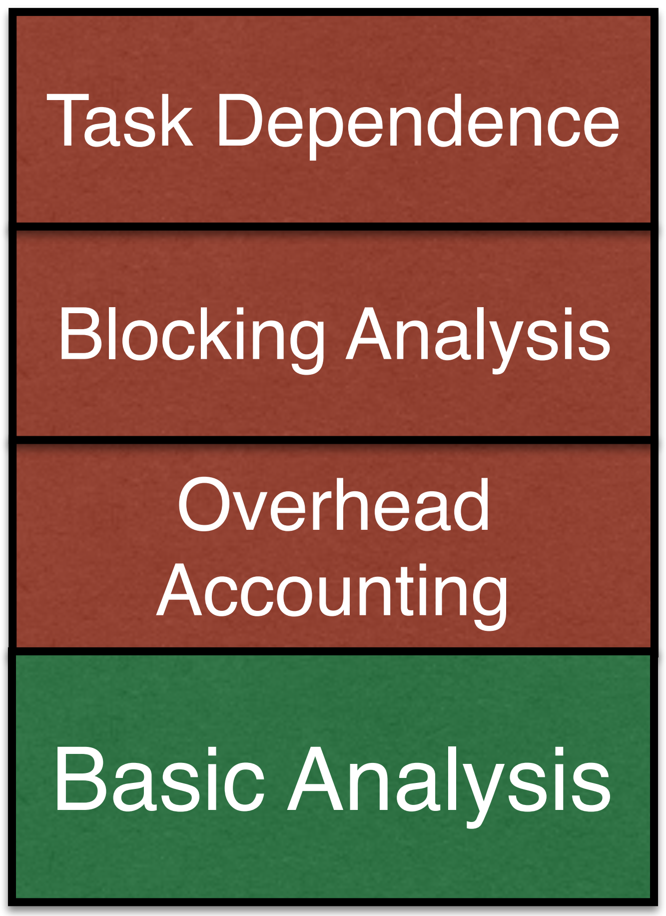
Safe Composition

Analyses sometimes need refining

Basic Analysis

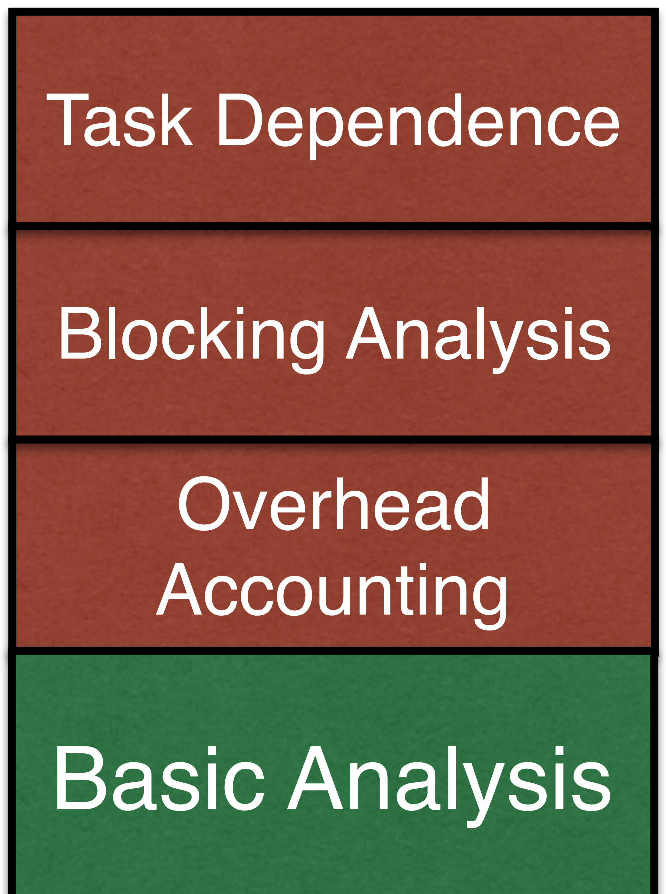
In most analyses, practical details
are assumed to be negligible.

Analyses sometimes need refining



But when deploying actual systems,
we might need to **refine the analysis**.

Analyses sometimes need refining



We call these extensions
(i.e., same results + tweaks)
neighboring proofs.

But when deploying actual systems,
we might need to **refine the analysis**.

Example: incorporating release jitter

	Basic RTA	RTA with Jitter
Uniprocessor	$R_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil e_j$	$r_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{r_i + J_j}{T_j} \right\rceil e_j$ $R_i = J_i + e_i + r_i$

It has been known for more than 20 years how to incorporate release jitter into uniprocessor RTA [3].

[3] Audsley, N., Burns A., Richardson, M., Tindell, K., and Wellings, A. “Applying new scheduling theory to static priority pre-emptive scheduling,” Software Engineering Journal, vol. 8, no. 5, pp. 284-292, 1993.

Example: incorporating release jitter

	Basic RTA	RTA with Jitter
Uniprocessor	$R_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil e_j$	$r_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{r_i + J_j}{T_j} \right\rceil e_j$ $R_i = J_i + e_i + r_i$
Multiprocessor	$R_i \leftarrow e_i + \frac{1}{m} \cdot \sum_{\tau_j \in hp_i} \left\lfloor \frac{I_j(R_i)}{T_j} \right\rfloor r_j$	**???**

But this result has not been proven for multiprocessor RTA.

Can we do the same for multiprocessors?

	Basic RTA	RTA with Jitter
Uniprocessor	$R_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil e_j$	$r_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{r_i + J_j}{T_j} \right\rceil e_j$ $R_i = J_i + e_i + r_i$
Multiprocessor	$R_i \leftarrow e_i + \frac{1}{m} \cdot \sum_{\tau_j \in hp_i} \left\lfloor \frac{I_j(R_i)}{T_j} \right\rfloor r_j$	**???**

Just sum up
the max jitter?

The answer is that we don’t know

Different system models have **different** assumptions.

What if changing the model **breaks some existing proof?**

Recent case: self-suspending tasks

Misuse of release jitter in the analysis caused bugs in 12 papers related to self-suspensions!

Excerpt from [1]:

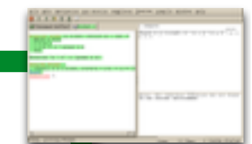
– Incorrect quantification of jitter for dynamic self-suspending task systems, which was used in [3,4,37,58]. This misconception was unfortunately adopted in [12, 14, 28, 36, 40, 73, 74, 76] to analyze the worst-case response time for partitioned multiprocessor real-time locking protocols.

[1] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, and D. de Niz, "Many suspensions, many problems: A review of self-suspending tasks in real-time systems," Department of Computer Science, TU Dortmund, Tech. Rep. 854, 2016

How to derive safe extensions?

How to derive safe extensions?

Mechanized proofs



Opportunity: neighboring proofs are conducted **systematically**.

We just need to refine the analysis and
let the proof assistant recheck the proofs.

If there is a bug, $\begin{cases} \text{it will always be detected.} \\ \text{we know exactly what to fix.} \end{cases}$

Why mechanized proofs?

Guaranteed Correctness

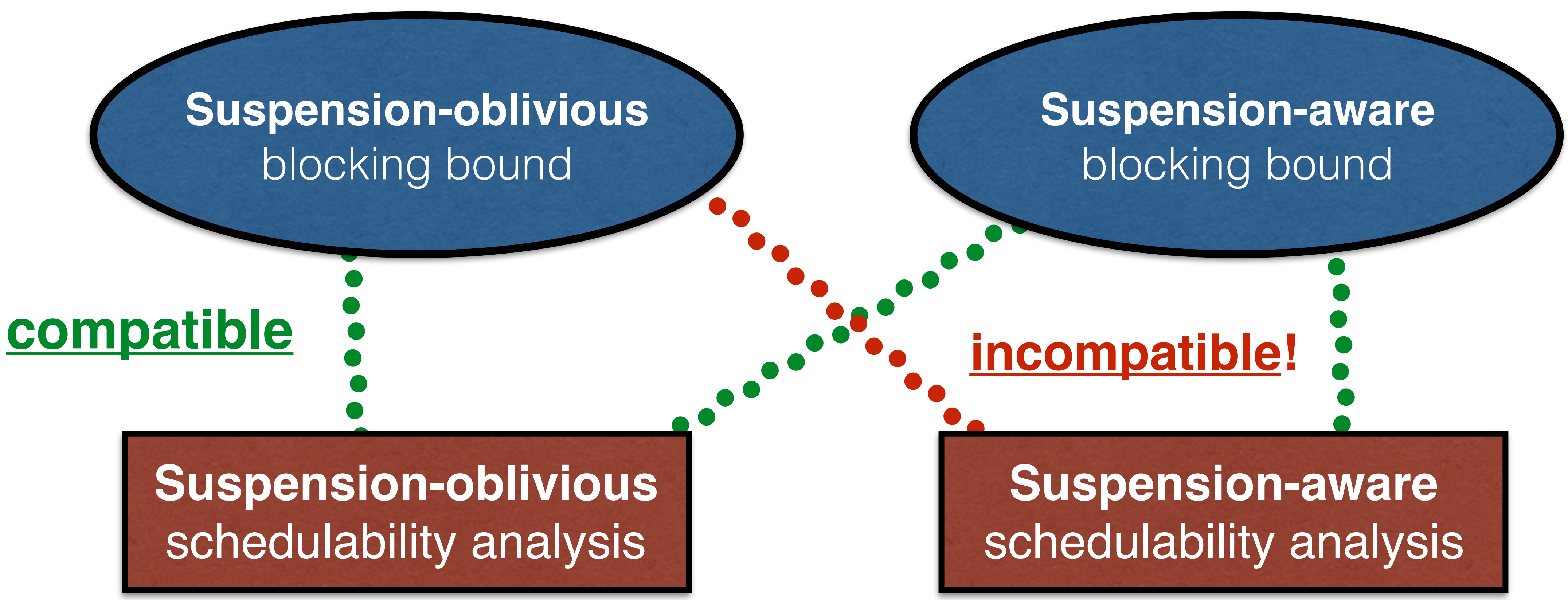
Trustworthy Extensions

Safe Composition

Sometimes we have to combine different analyses

Even if each analysis is **individually correct,**
they should not be combined if assumptions mismatch.

Example:



How to avoid mismatching assumptions?

How to avoid mismatching assumptions?

Mechanized proofs

Opportunity: mismatching assumptions are **automatically caught by the proof assistant.**

We just need to **avoid stating contradictory assumptions.**

But this can also be mechanically verified!

No more correctness pitfalls

Guaranteed Correctness

Trustworthy Extensions

Safe Composition

Mechanized proofs provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa

Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa

Verification has many challenges

Verification has many challenges

"Formal specifications are complex and full of symbols."

Verification has many challenges

"Formal specifications are complex and full of symbols."

"It might take many decades to verify all we know about real-time scheduling."

Verification has many challenges

"Formal specifications are complex and full of symbols."

"It might take many decades to verify all we know about real-time scheduling."

"Knowledge about formal methods tends to be restricted to few research groups."

Verification has many challenges

"Formal specifications are complex and full of symbols."

"It might take many decades to verify all we know about real-time scheduling."

"Knowledge about formal methods tends to be restricted to few research groups."

But there's an opportunity to change…

Principles & Goals of Prosa

1. **Readability** is crucial
2. Some proofs are **more important than others**
3. We should **maintain the proof culture**
4. **Community involvement**

Principle 1: Readability is crucial

The specification should be accessible to researchers with no previous experience with formal methods.

Principle 1: Readability is crucial

The specification should be accessible to researchers with no previous experience with formal methods.

We favor:

Many lemmas, short proofs (few dozen lines)

Long, verbose names and few cryptic symbols

Heavy use of documentation

Complex notation harms readability

Duration Calculus [Yuhua and Chaochen, 1994]

Furthermore, if there exists a ready task which is not running, then no processor should be idle. So let,

$$SCH_m \mathrel{\widehat{=}} \begin{pmatrix} \neg\Diamond \begin{pmatrix} true; Run(S) \\ \wedge(\bigvee_{i\in S}\bigvee_{j\in\alpha-S}(Urgt(j,i)\ ;\ \lceil p_j.rdy\rceil \wedge Run(S))) \end{pmatrix} \\ \wedge\Box(Run(S)\wedge\lceil\bigvee_{i\in\alpha-S} p_i.rdy\rceil \ \Rightarrow\ \sharp S = m) \end{pmatrix}$$

Complex notation harms readability

Duration Calculus [Yuhua and Chaochen, 1994]

Furthermore, if there exists a ready task which is not running, then no processor should be idle. So let,

$$SCH_m \mathrel{\widehat{=}} \left(\begin{array}{l} \neg\Diamond \left(\begin{array}{l} true; Run(S) \\ \wedge(\bigvee_{i\in S} \bigvee_{j\in\alpha-S}(Urgt(j,i) \; ; \; \lceil p_j.rdy \rceil \wedge Run(S))) \end{array} \right) \\ \wedge\Box(Run(S) \wedge \lceil \bigvee_{i\in\alpha-S} p_i.rdy \rceil \;\Rightarrow\; \sharp S = m) \end{array} \right)$$

Prosa

```
(* A scheduler is work-conserving iff all processors
   are busy (non-idle) whenever a job is backlogged. *)
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

A scheduler is work-conserving iff...

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

A scheduler is work-conserving iff...

...for every job j and time t...

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

A scheduler is work-conserving iff…

…for every job j and time t…

…if job j is backlogged at time t, …

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

A scheduler is work-conserving iff...

...for every job j and time t...

...if job j is backlogged at time t, ...

...then every processor cpu has a job j_other...

Long names and few symbols

```
Definition work_conserving :=
  ∀ j ∀ t,
    backlogged job_cost sched j t →
    ∀ cpu, ∃ j_other,
      scheduled_on sched j_other cpu t.
```

A scheduler is work-conserving iff...

...for every job j and time t...

...if job j is backlogged at time t, ...

...then every processor cpu has a job j_other...

...that is scheduled on cpu at time t.

Principle 2: Some proofs are more important than others

To make progress, we should focus on practical results.

Principle 2: Some proofs are more important than others

To make progress, we should focus on practical results.

We should formalize **recent analyses** and move towards **multiprocessor scheduling.**

Critical results should be proven first.
E.g., proving **analysis safety** is more important than termination, time complexity or optimality.

Principle 3: Maintain the proof culture

To ensure accessibility, we should reuse the established proof style of the real-time systems community.

Principle 3: Maintain the proof culture

To ensure accessibility, we should reuse the established proof style of the real-time systems community.

We avoid **complex logics** (e.g., temporal operators) and **advanced constructs** from the proof assistant (e.g., records, canonical structures, etc.).

We favor instead **first-order logic, lists, functions, basic arithmetic**.

Unusual notation discourages adoption

EDF Optimality in PPTL [Zhang, 2014]

Lemma 6. *If P_i overflows at $t = kT_i$, there is no idle time unit in $[(k-1)T_i, kT_i]$.*

That is,

$$Sch \supset \left(\bigcirc^{kT_i} (ac_i < C_i) \to \bigcirc^{t} \left(\bigvee_{j=1}^{m} r_j = 1 \right) \right) \quad (k-1)T_i < t < kT_i.$$

Prosa — Definition of Instantaneous Service

```
Definition service_at (t: time) :=
  \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.
```

LaTeX-like operators improve readability

Instantaneous Service

```
Definition service_at (t: time) :=
  \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.
```

LaTeX-like operators improve readability

Instantaneous Service

```
Definition service_at (t: time) :=
  \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.
```

Sum over each processor...

LaTeX-like operators improve readability

Instantaneous Service

```
Definition service_at (t: time) :=
  \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.
```

Sum over each processor...

...where job j is scheduled...

LaTeX-like operators improve readability

Instantaneous Service

```
Definition service_at (t: time) :=
  \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.
```

Sum over each processor...

...where job j is scheduled...

...of 1 (i.e., a count).

Principle 4: Community involvement

Vision: shared repository of real-time scheduling concepts and proofs.

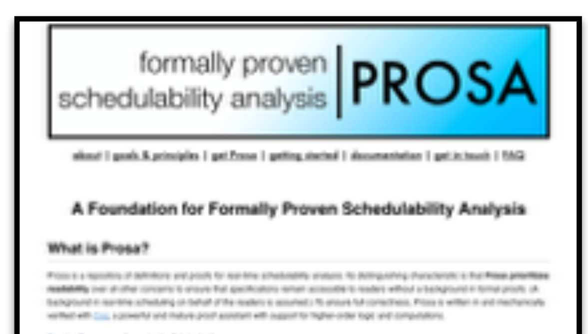
Principle 4: Community involvement

Vision: shared repository of real-time scheduling concepts and proofs.

We encourage **participation by the community**:

Specification accepted by the community + Mechanized Proofs = Non-disputable Results

Check out our website:
prosa.mpi-sws.org



Mechanized proofs can reach the community at large

1. **Readability** is crucial
2. Some proofs are **more important than others**
3. We should **maintain the proof culture**
4. **Community involvement**

By focusing on **readability** and by **maintaining the established proof culture,** mechanized proofs **can reach the community at large.**

Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa

Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa

Prosa is a collection of definitions, assumptions and theorems

Definitions

Assumptions

Theorems

Prosa covers **many concepts** from real-time scheduling

Definitions

```
Library schedule:  instantaneous service,
                   cumulative service,
                   job is pending, job is complete…

Library interference:  total interference,
                       per-task interference…

Library platform:  work conservation,
                   priority enforcement…
```

Assumptions

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.

Hypothesis H_enforces_FP_policy:
  enforces_FP_policy job_cost job_task sched higher_priority.

Hypothesis H_work_conserving: work_conserving job_cost sched.

Hypothesis H_sequential_jobs: sequential_jobs sched.
```

[…]

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.
```

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.
```

In any given schedule and for any given actual job execution costs, ...

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.
```

In any given schedule and for any given actual job execution costs, ...

...jobs do not execute after completion.

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.

    Definition completed_jobs_dont_execute :=
      ∀ j ∀ t,
        service sched j t ≤ job_cost j.
```

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.
```

```
Definition completed_jobs_dont_execute :=
  ∀ j ∀ t,
    service sched j t ≤ job_cost j.
```

For every job j at any time t,

Theorems

Assumptions can be **easily checked** (~10–15 in each analysis)

Definitions

Assumptions

```
Hypothesis H_completed_jobs_dont_execute:
  completed_jobs_dont_execute job_cost sched.
```

```
Definition completed_jobs_dont_execute :=
  ∀ j ∀ t,
    service sched j t ≤ job_cost j.
```

For every job j at any time t,

...the service received by j is no larger than its cost.

Theorems

Theorems are proven in small steps using **lemmas**

Definitions

Assumptions

Theorems

```
Theorem workload_bounded_by_W :
  workload_of tsk t1 (t1 + delta) ≤ workload_bound.
```

Theorems are proven in small steps using **lemmas**

Definitions

Assumptions

Theorems

We upper-bound the workload of a task...

```
Theorem workload_bounded_by_W :
  workload_of tsk t1 (t1 + delta) ≤ workload_bound.
```

Theorems are proven in small steps using **lemmas**

Definitions

Assumptions

Theorems

```
Lemma workload_bound_many_periods_in_between :
  job_arrival j_lst - job_arrival j_fst t ≥ num_mid_jobs.+1
                                              × (task_period tsk).
```

We upper-bound the workload of a task...

...based on the minimum distance between its first and last jobs in the interval.

```
Theorem workload_bounded_by_W :
  workload_of tsk t1 (t1 + delta) ≤ workload_bound.
```

Prosa covers many concepts and is well-documented

Sporadic Tasks

Job priorities

Job arrival constraints

Response time bounds

Interference

Deadline misses

Parallelism

	Definition/Let	Lemma/Theorem
Total	714	699

We use short, easy-to-understand definitions.

	Specification	Proofs	Comments
Lines	6661	17104	3442

1 comment for every 2 lines of spec!

What we have proven so far

(in ~8 person months)

What we have proven so far

(in ~8 person months)

- **Sporadic Task Model**
  - Workload-based interference bounds for work-conserving and EDF schedulers

What we have proven so far

(in ~8 person months)

- **Sporadic Task Model**
  - Workload-based interference bounds for work-conserving and EDF schedulers
  - Definition and proofs of correctness and termination of Bertogna and Cirinei's RTA for FP scheduling
    - ➡ Same for Bertogna and Cirinei's RTA for EDF scheduling

What we have proven so far

(in ~8 person months)

- **Sporadic Task Model**
  - Workload-based interference bounds for work-conserving and EDF schedulers
  - Definition and proofs of correctness and termination of Bertogna and Cirinei's RTA for FP scheduling
    - ➡ Same for Bertogna and Cirinei's RTA for EDF scheduling
  - Implementation of a work-conserving scheduler to test for contradictory assumptions

What we have proven so far

(in ~8 person months)

- **Sporadic Task Model**
  - Workload-based interference bounds for work-conserving and EDF schedulers
  - Definition and proofs of correctness and termination of Bertogna and Cirinei's RTA for FP scheduling
    - ➡ Same for Bertogna and Cirinei's RTA for EDF scheduling
  - Implementation of a work-conserving scheduler to test for contradictory assumptions
- **Extensions**
  - ➡ Same definitions and proofs for workloads with release jitter
  - ➡ Same definitions and proofs for workloads with parallel jobs

novel results

What we have proven so far

(in ~8 person months)

- **Sporadic Task Model**
  - Workload-based interference bounds for work-conserving and EDF schedulers
  - Definition and proofs of correctness and termination of Bertogna and Cirinei's

Thanks to **mature proof assistants and libraries**, Prosa allows mechanizing **recent and complex** schedulability analyses in **reasonable time.**

assumptions

- **Extensions**
  - ➡ Same definitions and proofs for workloads with release jitter
  - ➡ Same definitions and proofs for workloads with parallel jobs

Future Work

Future Work

1. Correct recently refuted proofs
   a) ~~APA scheduling~~ **(done! see prosa.mpi-sws.org/apa)**
   b) Self-suspending tasks

Future Work

1. Correct recently refuted proofs

   a) ~~APA scheduling~~ **(done! see prosa.mpi-sws.org/apa)**

   b) Self-suspending tasks

2. Verify practical results

   a) Semi-partitioned scheduling (e.g. C=D)

   b) Blocking analysis

   c) Overhead accounting

Future Work

1. Correct recently refuted proofs
   a) ~~APA scheduling~~ **(done! see prosa.mpi-sws.org/apa)**
   b) Self-suspending tasks
2. Verify practical results
   a) Semi-partitioned scheduling (e.g. C=D)
   b) Blocking analysis
   c) Overhead accounting
3. Investigate how to integrate Prosa with analysis tools and scheduler implementations

Disclaimer

Disclaimer

Not every proof has to be formalized.

Disclaimer

Not every proof has to be formalized.

Pen-and-paper proofs are **still useful.**

Disclaimer

Not every proof has to be formalized.

Pen-and-paper proofs are **still useful.**

We aim for **readable specifications,** but writing formal proofs **remains non-trivial.**

formally proven schedulability analysis | PROSA

More info at prosa.mpi-sws.org

Mechanized proofs provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

By focusing on **readability** and by **maintaining the established proof culture,** mechanized proofs **can reach the community at large.**

Thanks to **mature proof assistants and libraries**, Prosa allows mechanizing **recent and complex** schedulability analyses in **reasonable time.**

Backup slides

Generality of discrete time

Theorem 6 *A sporadic arbitrary-deadline task system $\mathcal{T}$ is feasible with respect to continuous schedules iff it is feasible with respect to discrete schedules.*

Results about dense time could still be formalized with Coq libraries for real numbers, e.g. Coquelicot.

[2] Bonifaci, V. and Marchetti-Spaccamela, A., "Feasibility analysis of sporadic real-time multiprocessor task systems," in Proc. of the 18th Annual European Symposium on Algorithms (ESA'10), 2010.

Working with Real Numbers

Coquelicot:

A User-Friendly Library of Real Analysis for Coq

Formalization of limits, continuity, differentiability, Riemann integrals, series, etc.

More info at coquelicot.saclay.inria.fr

Library: Probability Theory

Total/conditional probability, Bayes' theorem, random variables and finite distributions

```
Lemma prob_decomp: forall A B,
  \Pr_d[A] = \Pr_d[A :&: B] + \Pr_d[A :&: ~:B].
```

Moreira, D. *Finite Probability Distributions in Coq (2012).*

Related Work

Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Proof of EDF optimality *[Yuhua and Chaochen 1994]*
  — improved version *[Zhan 2000]*
- Schedulability condition of RM *[Schuzhen et al. 1999]*
- Simplified proofs and review *[Xu and Zhan 2008]*

Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Proof of EDF optimality *[Yuhua and Chaochen 1994]*
  — improved version *[Zhan 2000]*
- Schedulability condition of RM *[Schuzhen et al. 1999]*
- Simplified proofs and review *[Xu and Zhan 2008]*

+ Formalism reduces ambiguity

Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Proof of EDF optimality *[Yuhua and Chaochen 1994]*
  — improved version *[Zhan 2000]*
- Schedulability condition of RM *[Schuzhen et al. 1999]*
- Simplified proofs and review *[Xu and Zhan 2008]*

+ Formalism reduces ambiguity

- Complex logics and manual proofs
- Only uniprocessor scheduling

Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* *[Wilding 1998]*
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols *[Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]*
- Schedulability conditions based on **task phase** using *Coq* *[De Rauglaudre 2012]*
- Certified Computations of **Network Calculus** in Isabelle/HOL *[Mabille et al. 2013]*
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq *[Zhang et al. 2014]*

Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* *[Wilding 1998]*
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols *[Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]*
- Schedulability conditions based on **task phase** using *Coq* *[De Rauglaudre 2012]*
- Certified Computations of **Network Calculus** in Isabelle/HOL *[Mabille et al. 2013]*
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq *[Zhang et al. 2014]*

+ Mechanically-checked
- No results about multiprocessors

Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* *[Wilding 1998]*
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols *[Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]*
- Schedulability conditions based on **task phase** using *Coq* *[De Rauglaudre 2012]*
- Certified Computations of **Network Calculus** in Isabelle/HOL *[Mabille et al. 2013]*
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq *[Zhang et al. 2014]*

+ Mechanically-checked
- No results about multiprocessors
- Not widely adopted by our community

Model checking and timed automata

- Analysis of **uniprocessor** FP scheduling using UPPAAL
  [Fersman at al. 2006]
- Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks** using UPPAAL and NuSMV
  [Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]
- Analysis of **sporadic tasks** based on state exploration and automata reachability *[Baker and Cirinei 2007] [Geeraerts et al. 2012] [Burmyakov et al. 2015] [Sun and Lipari 2015]*

Model checking and timed automata

- Analysis of **uniprocessor** FP scheduling using UPPAAL *[Fersman at al. 2006]*
- Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks** using UPPAAL and NuSMV *[Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]*
- Analysis of **sporadic tasks** based on state exploration and automata reachability *[Baker and Cirinei 2007] [Geeraerts et al. 2012] [Burmyakov et al. 2015] [Sun and Lipari 2015]*

+ Multiprocessor, exact schedulability analysis

Model checking and timed automata

- Analysis of **uniprocessor** FP scheduling using UPPAAL
  [Fersman at al. 2006]
- Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks** using UPPAAL and NuSMV
  [Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]
- Analysis of **sporadic tasks** based on state exploration and automata reachability *[Baker and Cirinei 2007] [Geeraerts et al. 2012] [Burmyakov et al. 2015] [Sun and Lipari 2015]*

+ Multiprocessor, exact schedulability analysis
- State explosion (≤ 10 tasks or 4 processors)

Avoiding contradictory assumptions

1. **Implement a scheduler function S** using the proof assistant (take pending jobs, sort by priority, assign to CPUs, …).

2. **Prove that scheduler S satisfies every requirement** of the analysis (work-conserving, enforces priority, etc.) in an assumption-free context.

3. Since S is an actual algorithm, it is **impossible that two contradictory assumptions are satisfied by S.**