Contract-Based Integration of Cyber-Physical Analyses

Ivan Ruchkin
Dionisio De Niz
Sagar Chaki
David Garlan

October 14, 2014
14th International Conference on Embedded Software

Carnegie Mellon · institute for SOFTWARE RESEARCH · Software Engineering Institute

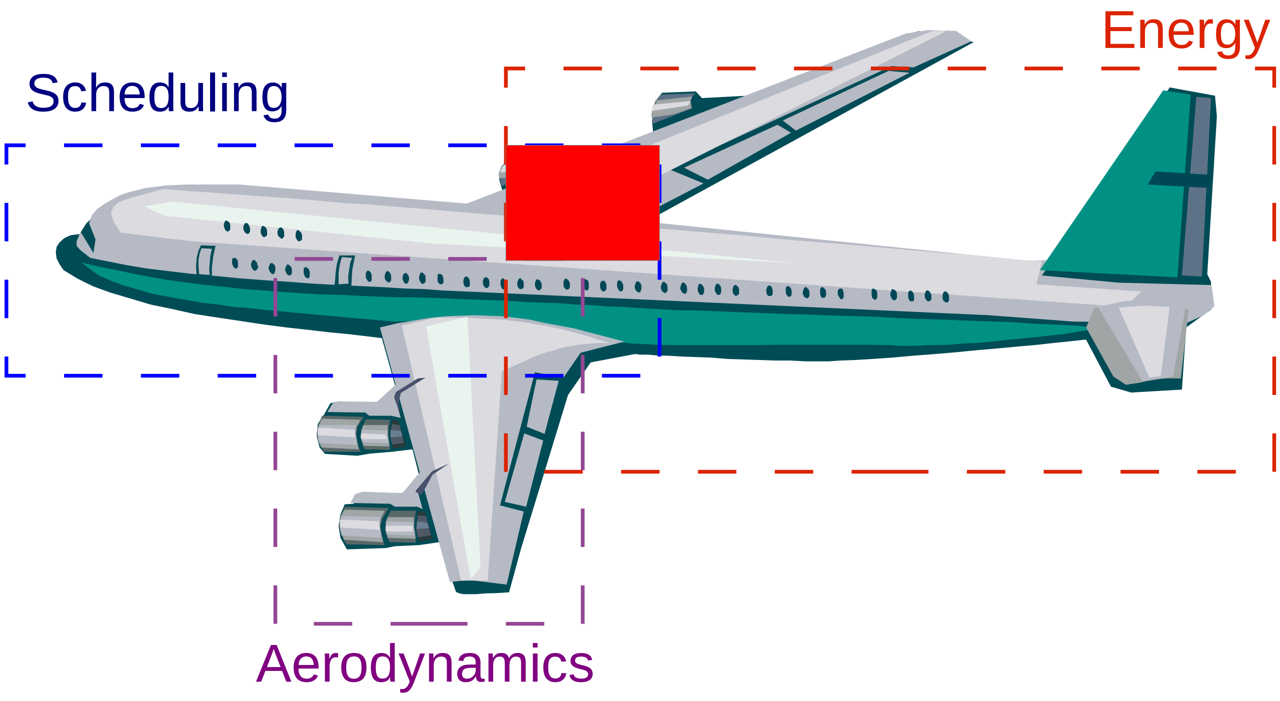
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0001714

Outline

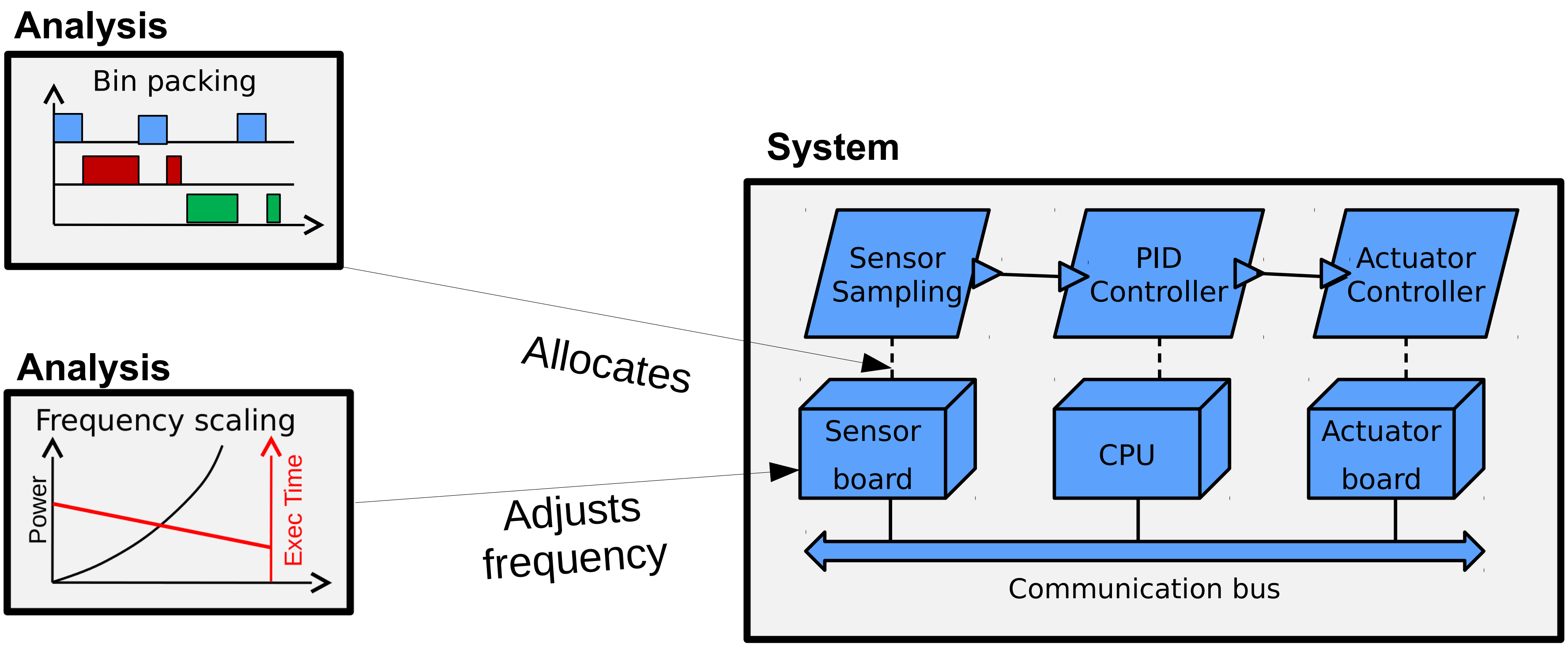
- Analysis integration problem
- Analysis contracts approach
  - Specification
  - Verification
- Experimental results

Model integration in CPS



- Subtle mismatches between technical domains
- Lead to costly fixes or failures

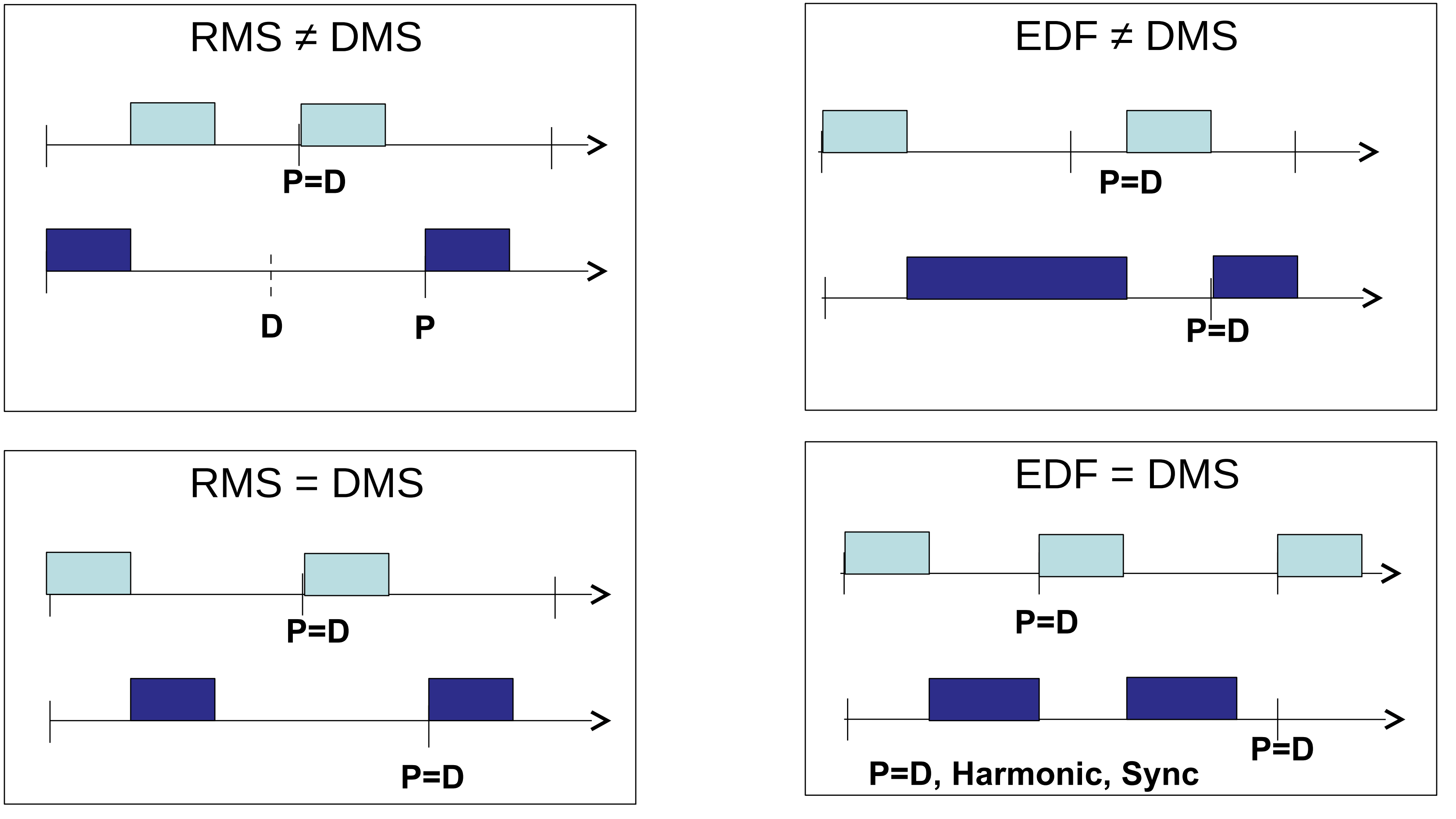
Analytic aspect of integration



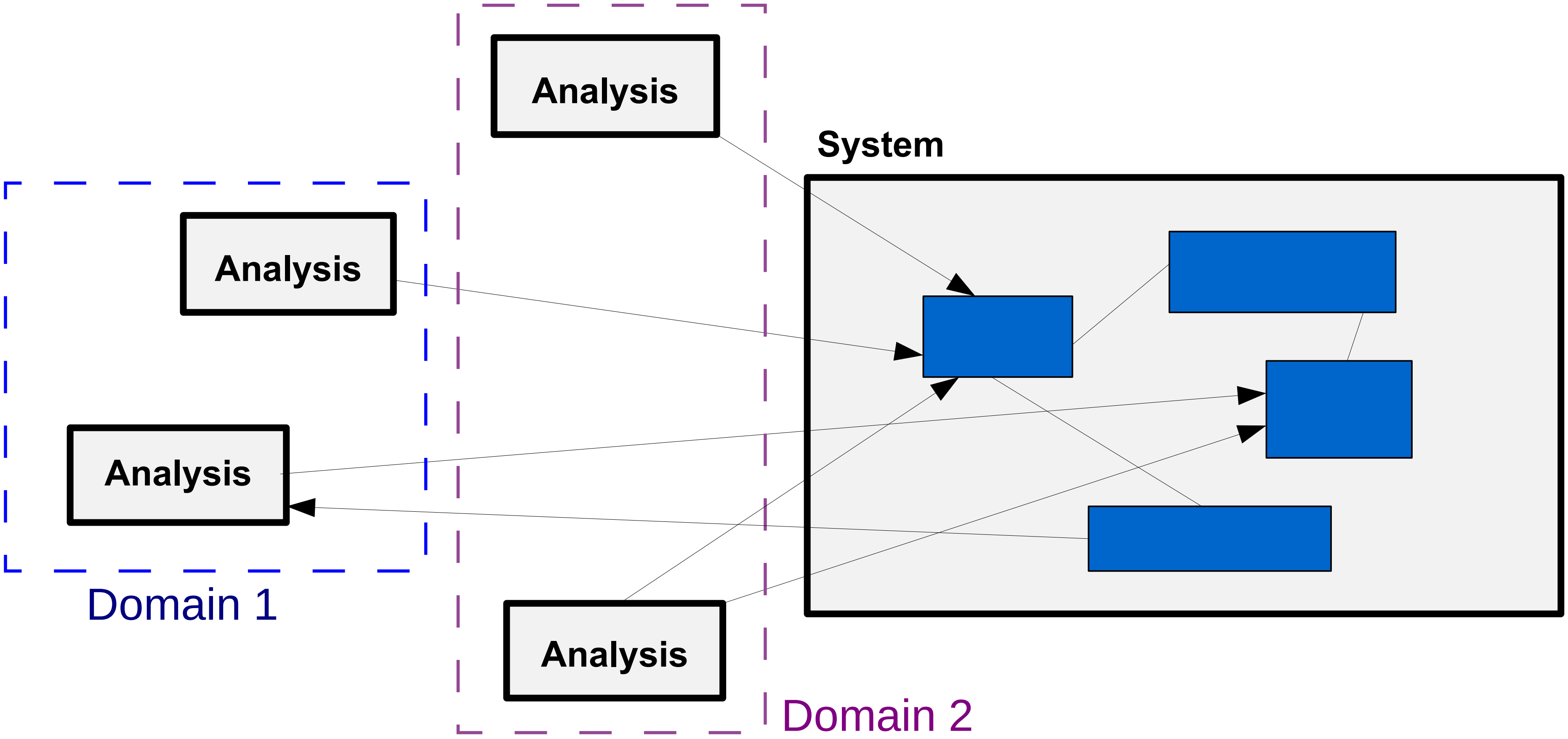
- Frequency scaling is applicable *only* when:
  - used after Bin packing
  - the system is behaviorally deadline-monotonic
- Otherwise, frequency scaling may render the system unschedulable

Frequency scaling assumption

- Behavioral equivalence to deadline-monotonic scheduling



Analysis integration problem



- Out-of-order execution
- Invalidation of assumptions

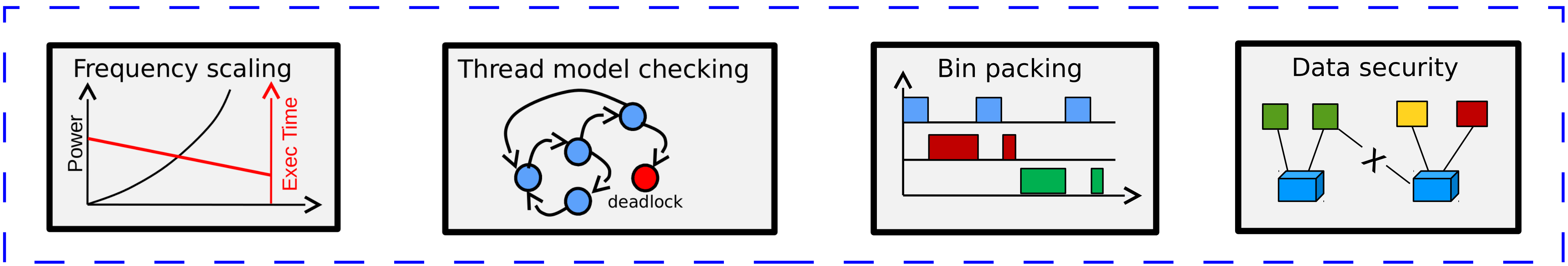
Existing solutions

- Assume-guarantee component composition does not handle analytic integration of tools [1][2].
- Architectural views tackle model consistency, not analytic tool consistency [3][4]
- Meta-level AADL languages do not allow domain-specific semantics [5]
- Previous work on analysis contracts: single domain only, unsound and incomplete verification [6]

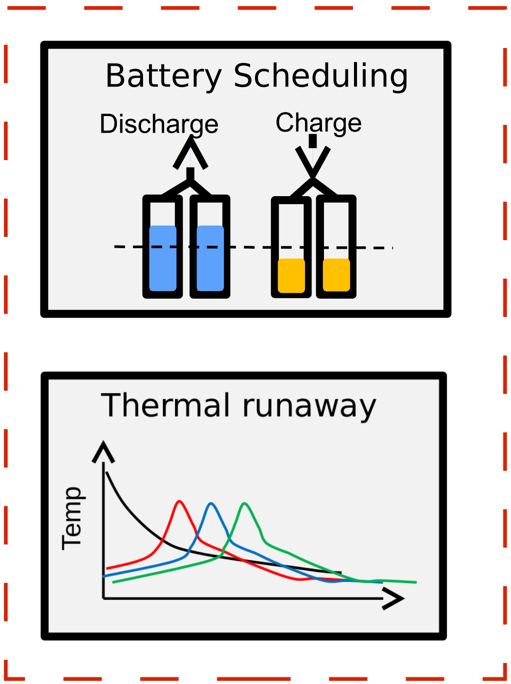
[1] Frehse et al. Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction, 2004
[2] Sangiovanni-Vincentelli et al. Taming Dr. Frankenstein: contract-based design for cyber-physical systems, 2013
[3] Torngren et al. Integrating viewpoints in the development of mechatronic products, 2013
[4] Rajhans et al. Supporting heterogeneity in cyber-physical systems architectures, 2014
[5] Boddy et al. The FUSED meta-language and tools for complex system engineering, 2011
[6] Nam et al. Resource allocation contracts for open analytic runtime models, 2011

Running example

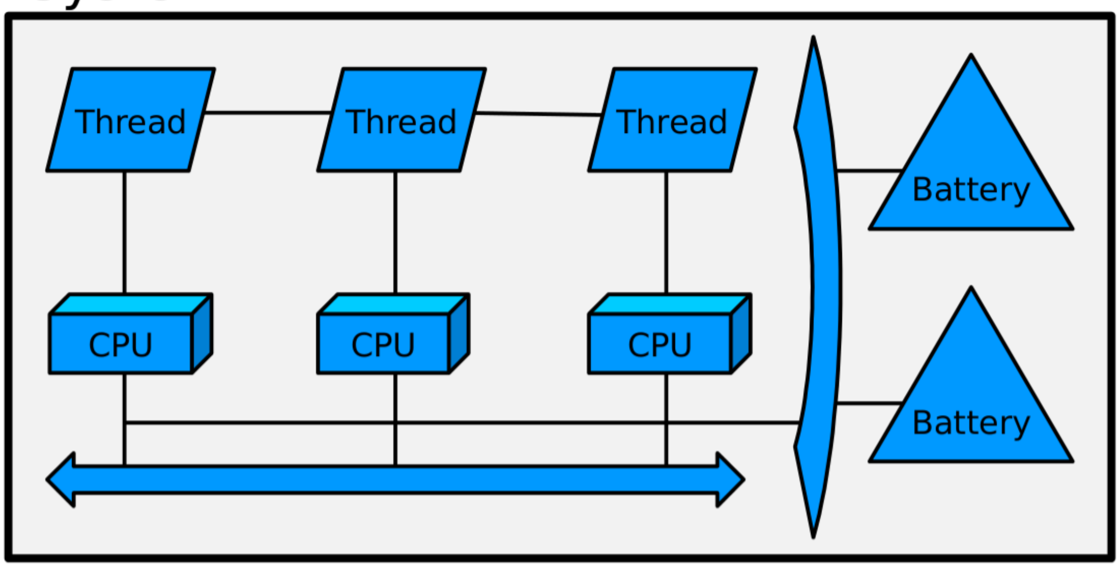
Scheduling



Battery



System



Outline

- Analysis integration problem
- **Analysis contracts approach**
  - Specification
  - Verification
- Experimental results

Analysis contracts approach

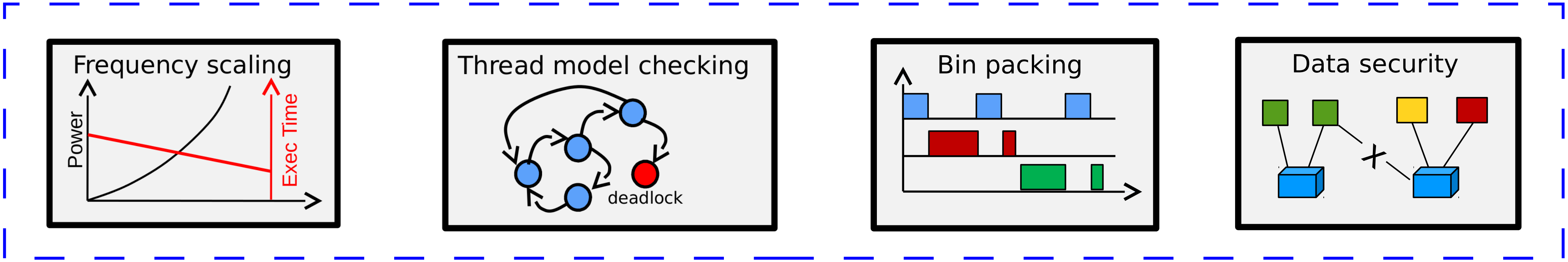
1. Formalize analysis domains
2. Specify dependencies, assumptions, and guarantees of analyses
3. Determine correct ordering of analyses
4. Verify assumptions and guarantees of analyses

Outline

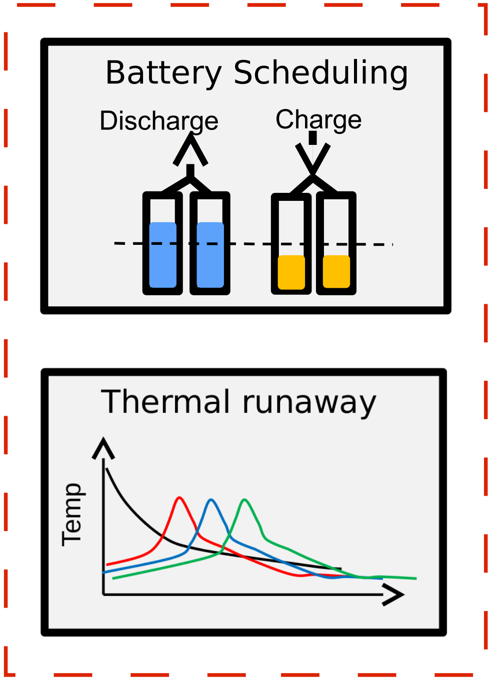
- Analysis integration problem
- Analysis contracts approach
  - **Specification**
  - Verification
- Experimental results

Running example

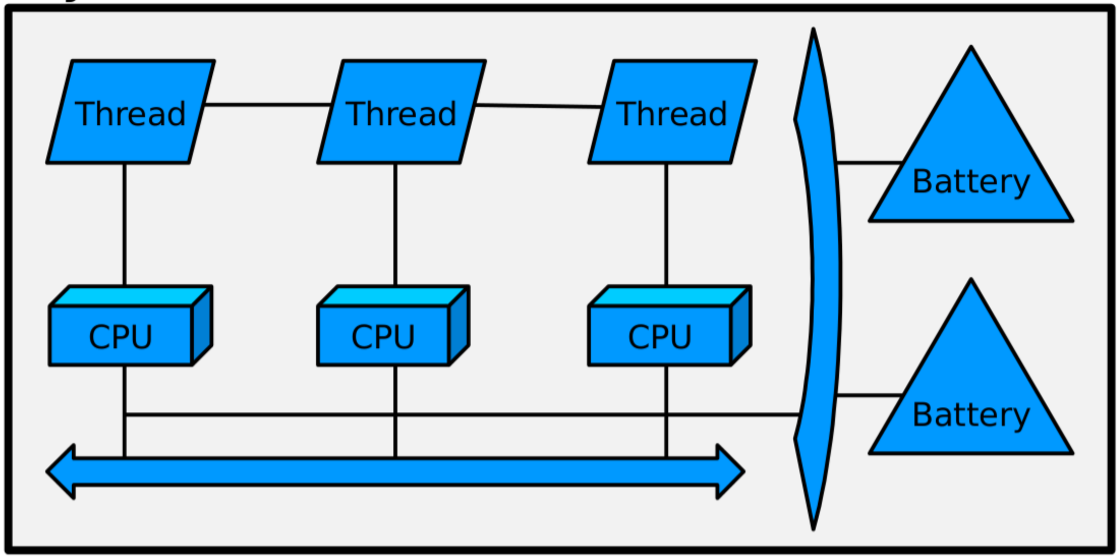
Scheduling



Battery

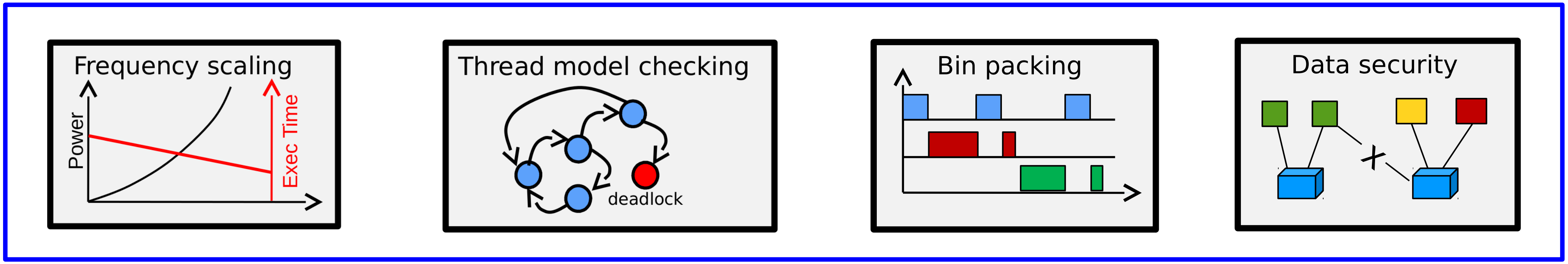


System

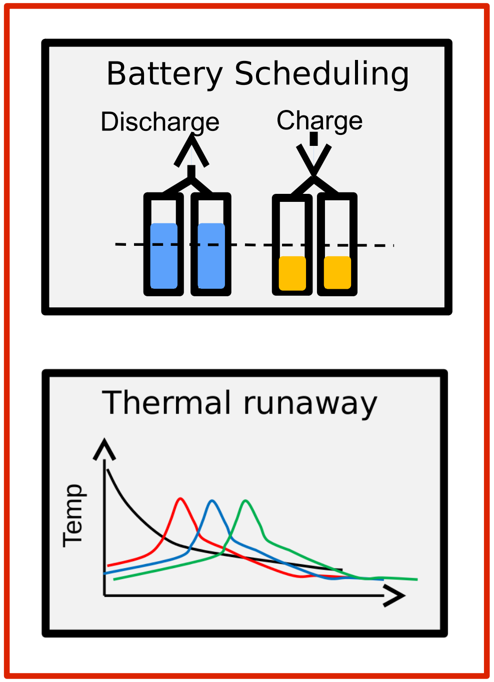


Running example

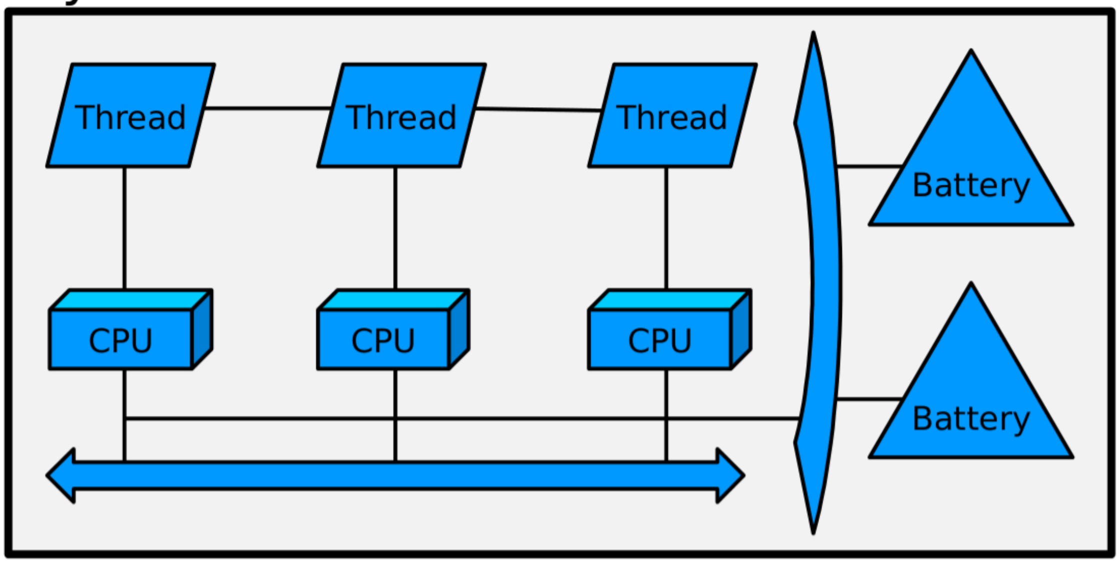
Scheduling domain $\boldsymbol{\sigma}_{\textbf{sched}}$



Battery domain $\boldsymbol{\sigma}_{\textbf{batt}}$



System



Verification domain

Domain **σ** is a many-sorted signature ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, $\mathcal{T}$, $⟦\,⟧_{\boldsymbol{\sigma}}$):

- $\mathcal{A}$ **–** set of atoms: $\mathbb{B}$, $\mathbb{Z}$, *Threads*, *Batteries, SchedPol*
- $\mathcal{S}$ **–** static functions: *Period, Dline, CPUBind, Voltage*
- $\mathcal{R}$ **–** runtime functions C*anPrmpt*: *Threads x Threads* → $\mathbb{B}$
- $\mathcal{T}$ **–** execution semantics
  - set of sequences of $\mathcal{R}$ assignments
- $⟦\,⟧_{\boldsymbol{\sigma}}$ and $⟦\,⟧_{\mathbf{m}}$ **–** domain and model interpretations
  - $⟦\mathit{SchedPol}⟧_{\boldsymbol{\sigma}}$ = {RMS, DMS, EDF}
  - $⟦\mathrm{CPUBind}⟧_{\mathbf{m}}$ **=** { ($Ctrl_1$, CPU_1), (Ctrl_2, CPU_2), ... }

Analysis contract

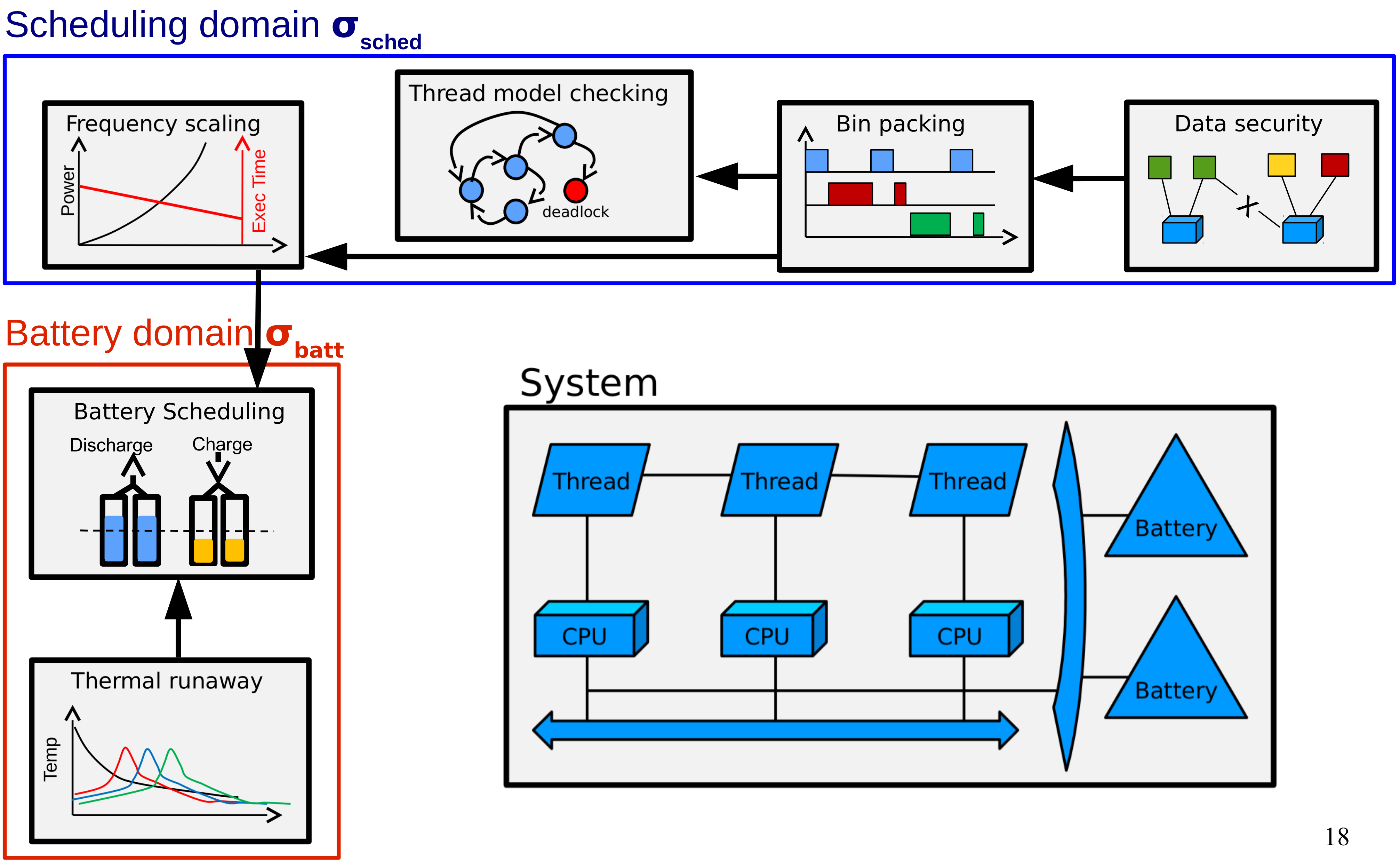
- Given a domain **σ**, *analysis contract* **C** is a tuple (**I**, **O**, **A**, **G**)
  - Inputs $\mathbf{I} \subseteq \mathcal{A} \cup \mathcal{S}$
  - Outputs $\mathbf{O} \subseteq \mathcal{A} \cup \mathcal{S}$
  - Assumptions $\mathbf{A} \subseteq \mathcal{F}_{\boldsymbol{\sigma}}$
  - Guarantees $\mathbf{G} \subseteq \mathcal{F}_{\boldsymbol{\sigma}}$
- Where:
  - $\mathcal{F}_{\boldsymbol{\sigma}}$ ::= $\{\forall|\exists\}\ v_1..v_n \bullet \varphi \ \ | \ \ \{\forall|\exists\}\ v_1..v_n \bullet \varphi : \psi$
  - φ is a static predicate formula over $\mathcal{A}$ and $\mathcal{S}$
  - ψ is an LTL formula over $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{R}$
  - E.g.: $\forall\ t_1, t_2$: *Threads* • $t_1 \neq t_2 \wedge$ *CPUBind*(t_1) = *CPUBind*(t_2) :

    G (*CanPrmpt*(t_1, t_2) ⇒ *Dline*(t_1) < *Dline*(t_2))

Outline

- Analysis integration problem
- Analysis contracts approach
  - Specification
  - **Verification**
- Experimental results

Running example



Assumption verification

- Goal:

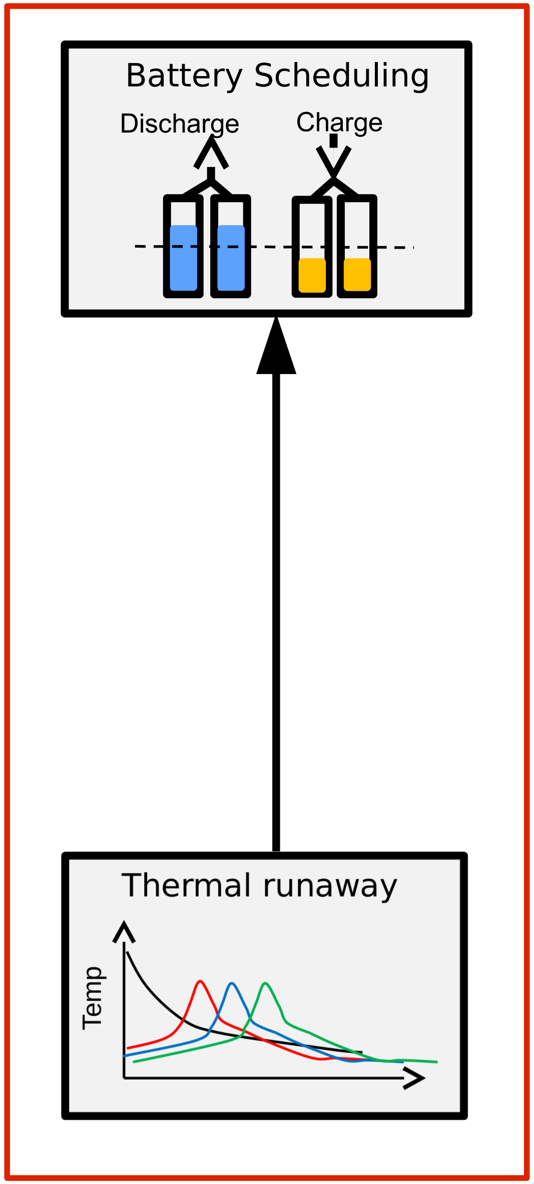
  $\forall\ t_1, t_2$: *Threads* • $t_1 \neq t_2 \wedge CPUBind(t_1) = CPUBind(t_2)$:

  $G\ (CanPrmpt(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2))$

- SMT solver finds solutions for static fragment φ
  - $\forall\ t_1, t_2$:*Threads* | $t_1 \neq t_2 \wedge CPUBind(t_1) = CPUBind(t_2)$
- Model checking property ψ in a behavioral Promela model for each SMT solution:
  - $G\ (CanPrmpt(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2))$

Battery modeling

Battery domain $\boldsymbol{\sigma}_{\mathbf{batt}}$

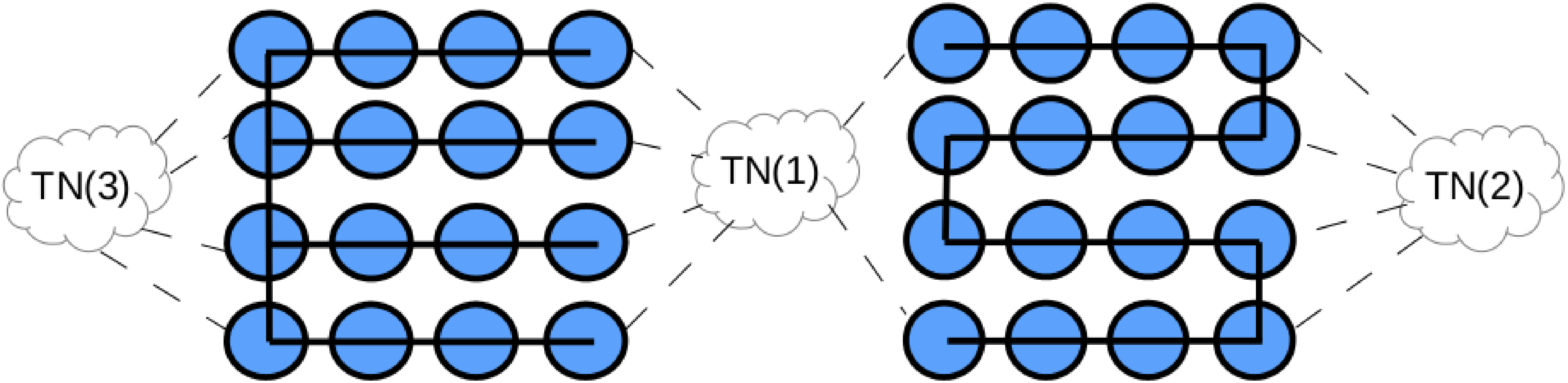


- Abstraction: circuits
- Selects a scheduler for cell connections
- Oblivious of heat: treats any configuration as acceptable heat-wise

- Restrictions on acceptable thermal configurations
- Guarantee: unacceptable ones don't occur

- Abstraction: geometry
- Simulates heat propagation
- Cannot scale to dynamic scheduling: simulates only fixed cell configurations

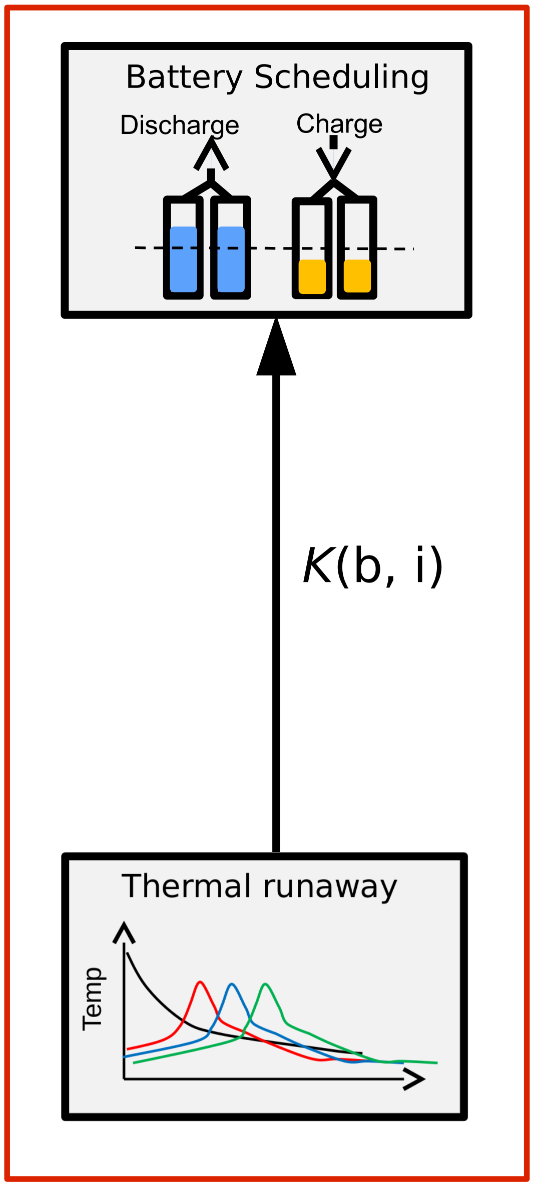
Battery scheduling guarantee



- ***G**: "Bad thermal configurations are not reachable"*
- *TN*(b, i) ∈ $\mathcal{R}$ – number of cells in b with i thermal neighbors
- *K*(b, i) ∈ $\mathcal{S}$ – experimental weight for *TN*(b, i)
- **G** = {∀ b: *Batteries* • G ($\sum_{i=0..3}$ *K*(b, i)**TN*(b, i)) ≥ 0}

Battery modeling

Battery domain σ_{batt}



Selects a battery scheduler

G: $\forall$ b: *Batteries* • G ($\sum_{i=0..3} K(b, i) * TN(b, i)$) ≥ 0

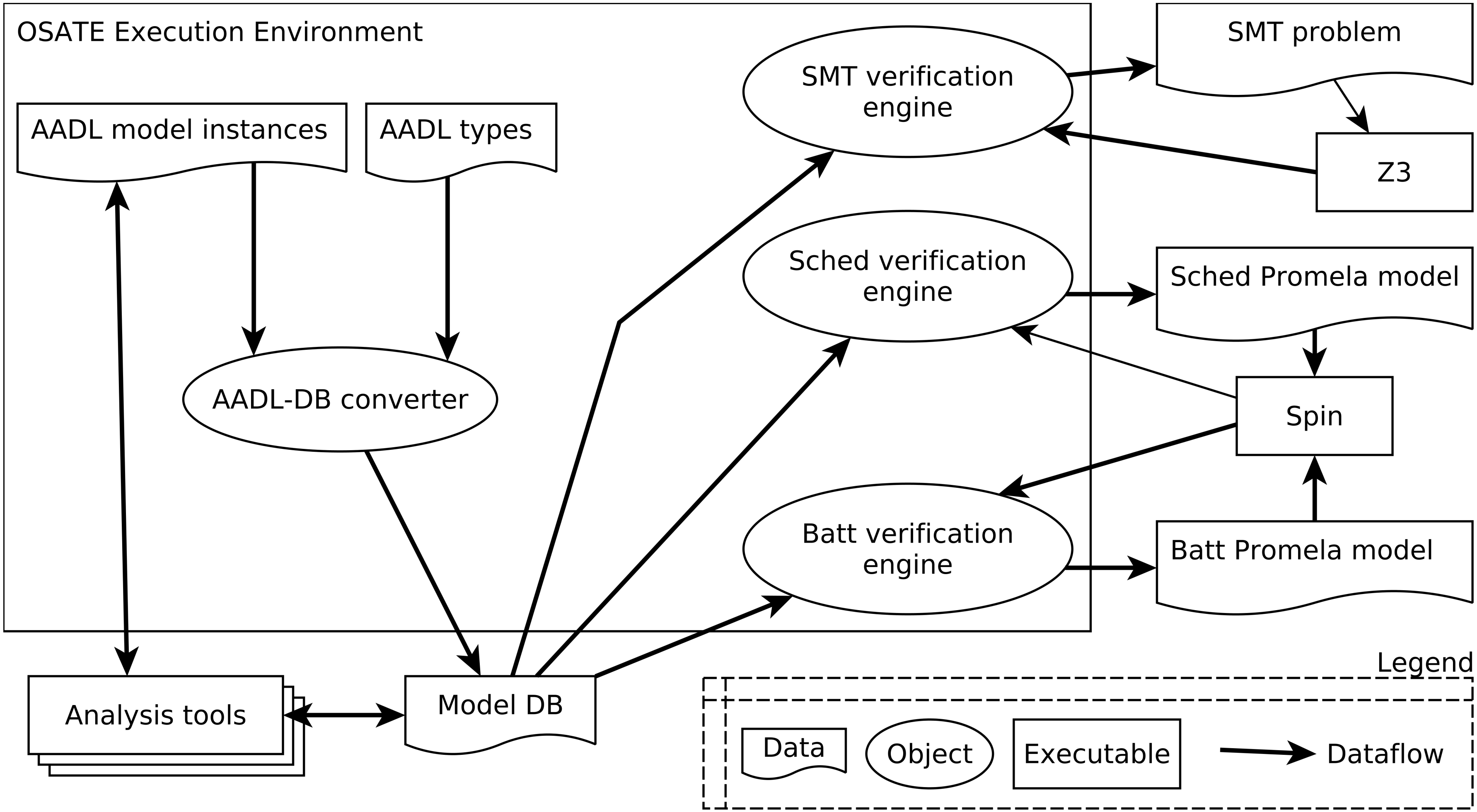
Verified with battery Promela/Spin model

Determines *K*(b, i) via simulation

Outline

- Analysis integration problem
- Analysis contracts approach
  - Specification
  - Verification
- **Experimental results**

Framework implementation



Scalability evaluation

- SMT solving typically takes less than 0.1 second
- Spin model checking times:

σ_{sched}:

Threads	(R/D)MS time	EDF time
3	0.01	0.01
4	0.01	0.52
5	0.07	33.4
6	0.37	2290.0
7	2.18	Out Mem
8	12.4	Out Mem
9	71.2	Out Mem
10	421	Out Mem
11	Out Mem	Out Mem

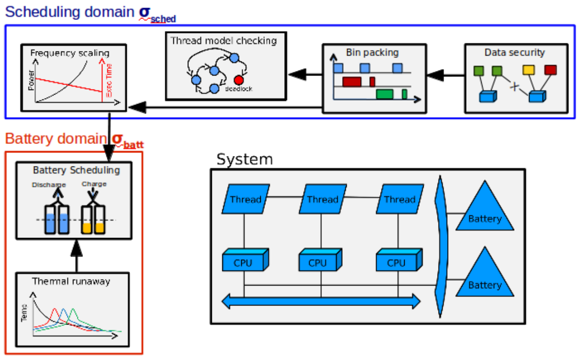
σ_{batt}:

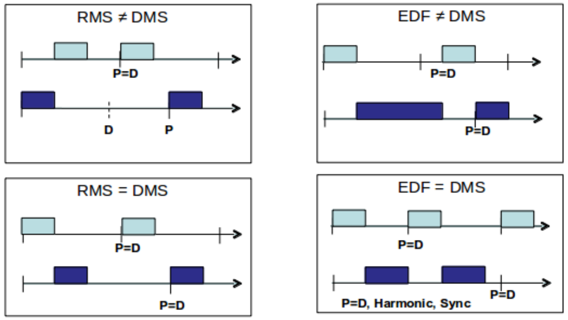
Cells	FGURR time	FGWRR time	GPWRR time
9	0.13	0.15	0.15
12	0.61	2.34	3.94
16	44	31.4	127
20	1060	619	Out Mem
25	Out Mem	Out Mem	Out Mem

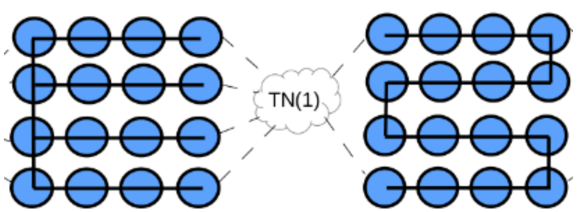
All times are in seconds

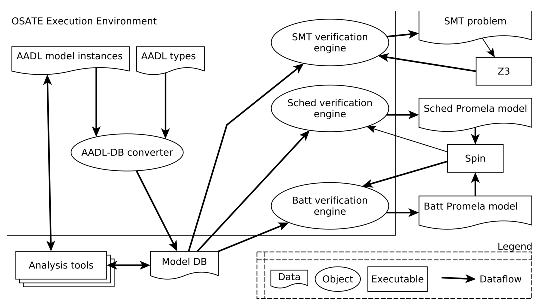
Summary

- Analysis integration is error-prone
  - Incorrect ordering
  - Violation of implicit assumptions
- Our solution:
  - Contract specification language
  - Contract verification algorithm
  - Framework implementation
- Effective, extensible, and scalable









Verification domain

- Domain **σ** is a many-sorted signature ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, $\mathcal{T}$, $\{\!\![\ ]\!\!\}_\sigma$):
  - $\mathcal{A}$: set of sorts – system elements and standard sorts
    - E.g.: $\mathbb{B}$, $\mathbb{Z}$, *Threads*, *Batteries, SchedPol*
  - $\mathcal{S}$: $\mathcal{A}_i$ x ... x $\mathcal{A}_n \rightarrow \mathcal{A}_k$ – static functions that encode design properties
    - E.g.: *Period, Dline, CPUBind, Voltage*
  - $\mathcal{R}$: $\mathcal{A}_i$ x ... x $\mathcal{A}_n \rightarrow \mathcal{A}_k$ – runtime functions that encode dynamic properties
    - E.g.: *CanPrmpt*: *Threads* x *Threads* → $\mathbb{B}$
      TN: *Batteries* x $\mathbb{Z} \rightarrow \mathbb{Z}$

Verification domain

- Domain **σ** is a many-sorted signature ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, $\mathcal{T}$, $⟦\,⟧_{\boldsymbol{\sigma}}$):
  - $\mathcal{T}$: execution semantics – set of sequences of $\mathcal{R}$ assignments
    - E.g.: thread scheduler state model for $\boldsymbol{\sigma}_{\mathbf{sched}}$
      battery state model for for $\boldsymbol{\sigma}_{\mathbf{batt}}$
  - $⟦\,⟧_{\boldsymbol{\sigma}}$: domain interpretation for $\mathcal{A}$ and $\mathcal{S}$
    - E.g.: $⟦\mathit{SchedPol}⟧_{\boldsymbol{\sigma}}$ = {RMS, DMS, EDF}
- Architectural model **m** is an interpretation $⟦\,⟧_{\mathbf{m}}$ of $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{T}$
  - E.g.: $⟦\mathit{Threads}⟧_{\mathbf{m}}$ = { SensorSample, $Ctrl_1$, $Ctrl_2$ }
    $⟦CPUBind⟧_{\mathbf{m}}$ **=** { ($Ctrl_1$, CPU_1), ($Ctrl_2$, CPU_2), … }
  - $⟦\,⟧_{\boldsymbol{\sigma}} \cup ⟦\,⟧_{\mathbf{m}}$ is a full interpretation

Contracts

Security Analysis

- $An_{sec}.C: I = \{T, ThSecCl\}, O = \{NotColoc\}, A = \emptyset, G = \{g\}$
  - $g: \forall\, t_1, t_2 \cdot ThSecCl(t_1) \neq ThSecCl(t_2) \Rightarrow t_1 \in NotColoc(t_2)$

Multiprocessor scheduling: (Binpacking + scheduling)

- $An_{sched}.C: I = \{T, C, NotColoc, Per, WCET, Dline\}, O = \{CPUBind\}, A = \emptyset, G = \{g\}$
  - $g: \forall\, t_1, t_2 \cdot t_1 \in NotColoc(t_2) \Rightarrow CPUBind(t_1) \neq CPUBind(t_2)$

Frequency Scaling

- $An_{freqsc}.C: I = \{T, C, CPUBind, Dline\}, O = \{CPUFreq\}, G = \emptyset, A = \{a\}$
  - $a: \forall t_1, t_2 \cdot CPUBind(t_1) = CPUBind(t_2): G(CanPrmpt(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2)$

Model checking periodic program (REK):

- $An_{rek}.C: I = \{T, C, Per, Dline, WCET, CPUBind\}, O = \{ThSafe\}, G = \emptyset, A = \{a_1, a_2\}$
- $a_1: \forall t \cdot Per(t) = Dline(t),\ \ a_2: \forall t_1, t_2 \cdot G(Canprmpt(t_1, t_2) \Rightarrow G\ \neg CanPrmpt(t_2, t_1))$

Thermal runaway:

- $An_{therm}.C: I = \{B, BatRows, BatCols, Voltage\}, O = \{K\}, A = \emptyset, G = \emptyset$

Battery Scheduling

- $An_{bsched}.C: I = \{B, BatRows, BatCols\}, O = \{BatConnSchedPol, HasReqLifetime, SeriqlReq, ParalRea\}, A = \emptyset, G = \{g\}$
- $g: G(K(0) \times TN(0) + K(1) \times TN(1) + K(2) \times TN(2) + K(3) \times TN(3) \geq 0)$