Scalable (yet Precise) Timing Analysis:
Of Course Model-Based!

Can **P** finish its execution within **D** sec's?

Wang Yi

Uppsala University

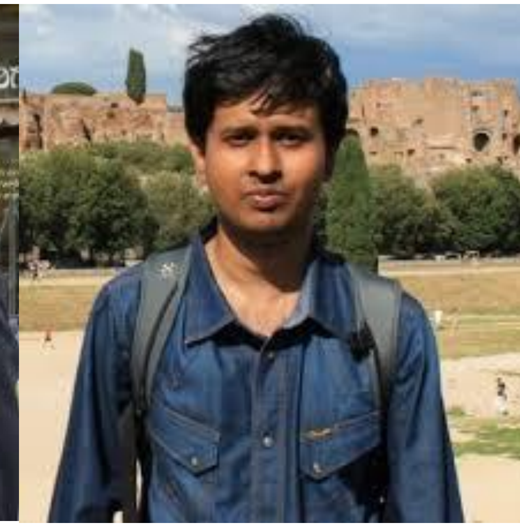
(ETAPS 2015, London)

Joint work with my students:

Nan Guan

Martin Stigge

Pontus Ekberg

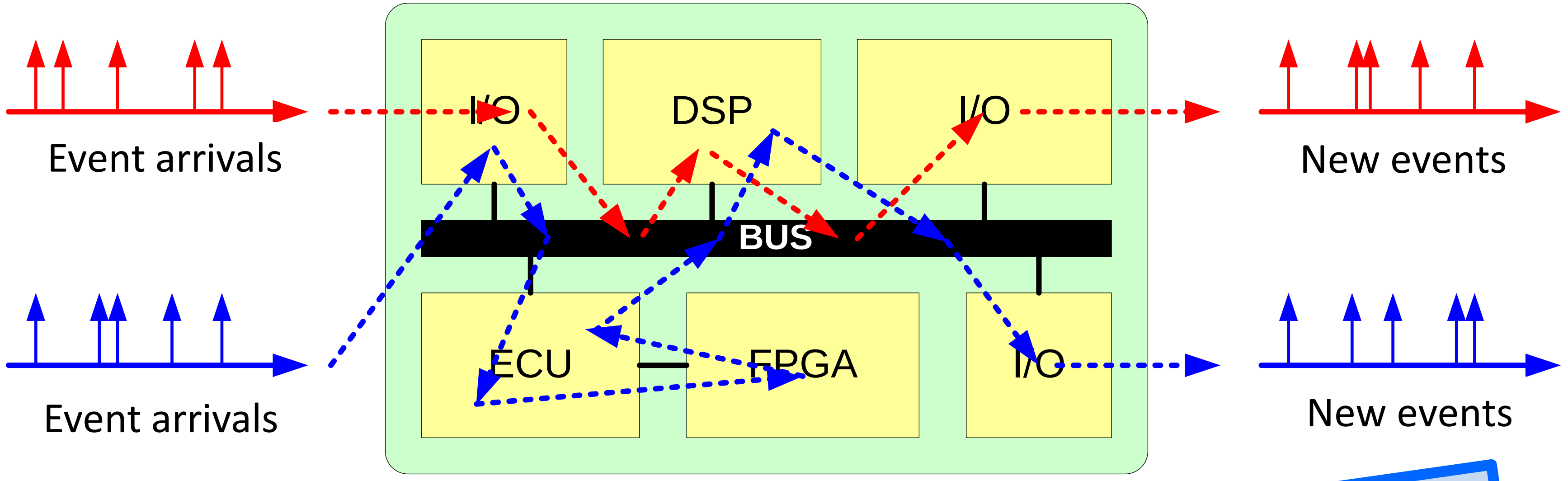


Jakaria Abdullah

OUTLINE

- Modeling with graph-based models
- Scalable Analysis (pseudo-polynomial time)
  - for the tractable cases
- Efficient Analysis (combinatorial refinement)
  - for the intractable cases

Embedded Systems



Timing Analysis

Example size of applications:
80-100 tasks (Ind. Robots)
30-50 tasks (FMS, THALES)

- **What is the maximal delay at each component?**
- **What is the maximal end-to-end delay?**

TACAS, Aarhus, April 1995

UPPAAL

Johan Bengtsson
Kim Larsen
Fredrik Larsson
Paul Pettersson
Wang Yi

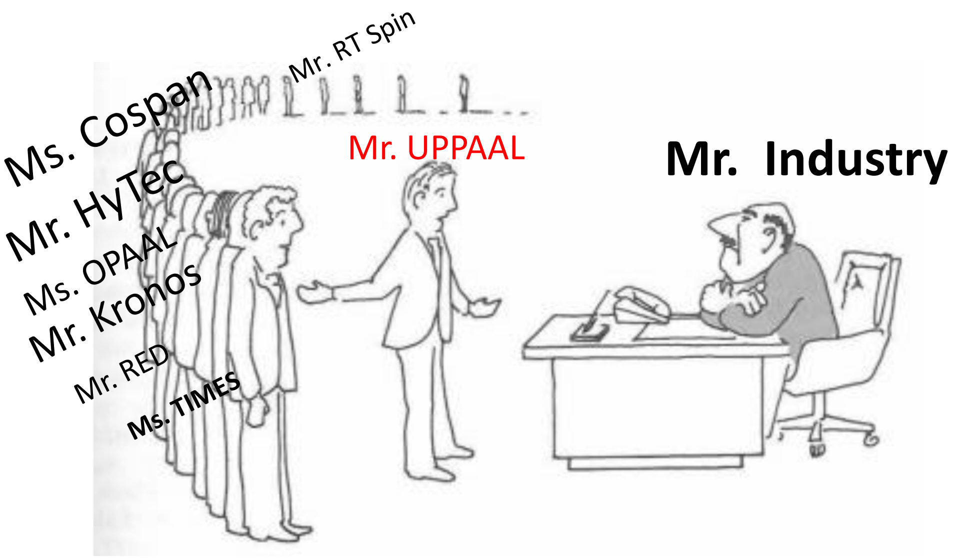
Photo: Kim Larsen, Aalborg Univ.

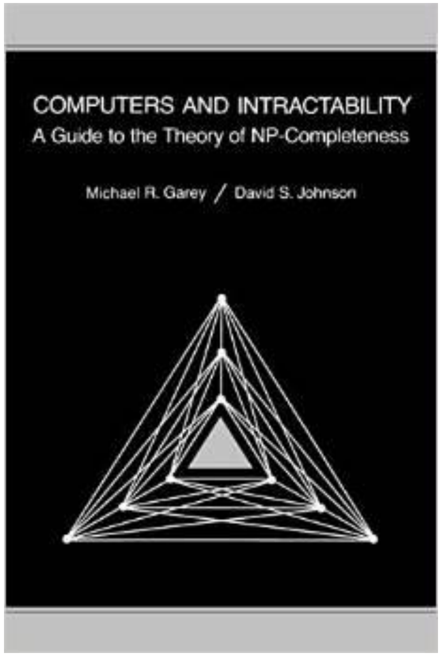
Model Checking of Timed Systems

- UPPAAL, UPPAAL-Tiga (FUSC): Timed automata (Uppsala Univ., Aaalborg Univ.)
  - manipulate UPPAAL XML (GPL-3, Python)
  - Yggdrasil (?, ?): UML (subset) -> Uppaal, intended for test generation (Aaalborg Univ.)
  - METAMOC (GPL-3, Python): WCET Analysis of ARM Processors using Real-Time model checking (Aalborg Univ.)
  - SARTS (?, Java): Model Based Schedulability Analysis of Real-Time Systems, SCJ2, UPPAAL (Aaalborg Univ.)
- OPAAL (GPL-3, Python): distributed/parallel (discrete time) model checker for networks of timed automata using MPI
- ECDAR (FUSC): timed interface theory (Aaalborg, INRIA, ITU)
- PyECDAR (GPL-2, Python): solve timed games based on timed automata models (ITU)
- IOA (MIT, Java): I/O automata formal language (MIT)
- TEMPO (closed, Java): Formal language for modeling distributed systems w/ | w/o timing constraints as collections of interacting state machines, i.e., timed input/output automata (TIOA) (UIUC)
  - Tempo2HSal (?, Python): Tempo (`.tioa`) -> HybridSAL (`.hsal`) translator (SRI)
- ATAS (GPL-3, Python): Alternating 1-clock (fully decidable) Timed Automata Solver
- PPL binding (GPL-2, Python): for Parma Polyhedral Lib features some specific methods for Timed Automata analysis
- MCPTA (FUSC, ?): Probabilistic Timed Automata model checker for MoDeST | UPPAAL | PRISM - maps on PRISM (Saarland Univ.)
- SAAtRE (?): Abstraction refinement model checker for Timed Automata based on extended SAT-solving, UPPAAL-like input format (Univ. Oldenburg, CWI)
- Fortuna (GPL-3, C++/Eclipse): MC priced probabilistic timed automata (PPTAs) (Univ. Twente)
- COSPAN (?, ?): Automata-theoretic verification of coordinating processes with timing constraints (UPenn)
- Romeo: timed Petri nets (IRCCyN)
- ExSched: develop operating system schedulers for VxWorks and Linux w/o modifying the underlying kernel ([Malardalen Univ.]http://www.es.mdh.se/staff/197-Mikael__sberg))
- RTComposer (Java): classes and utilities for predictable real-time scheduling (BenGurion, UPenn)
- ASTRAL: MC of real-time systems (UCSB)
- PAT (?, C#): simulator, MC, refinement checker for concurrent and RT systems (Nanyang Tech. Univ.)
- HCMC (? , C++): Compositional model checking for real-time systems (ENS-Cachan)

Could these tools solve this problem?

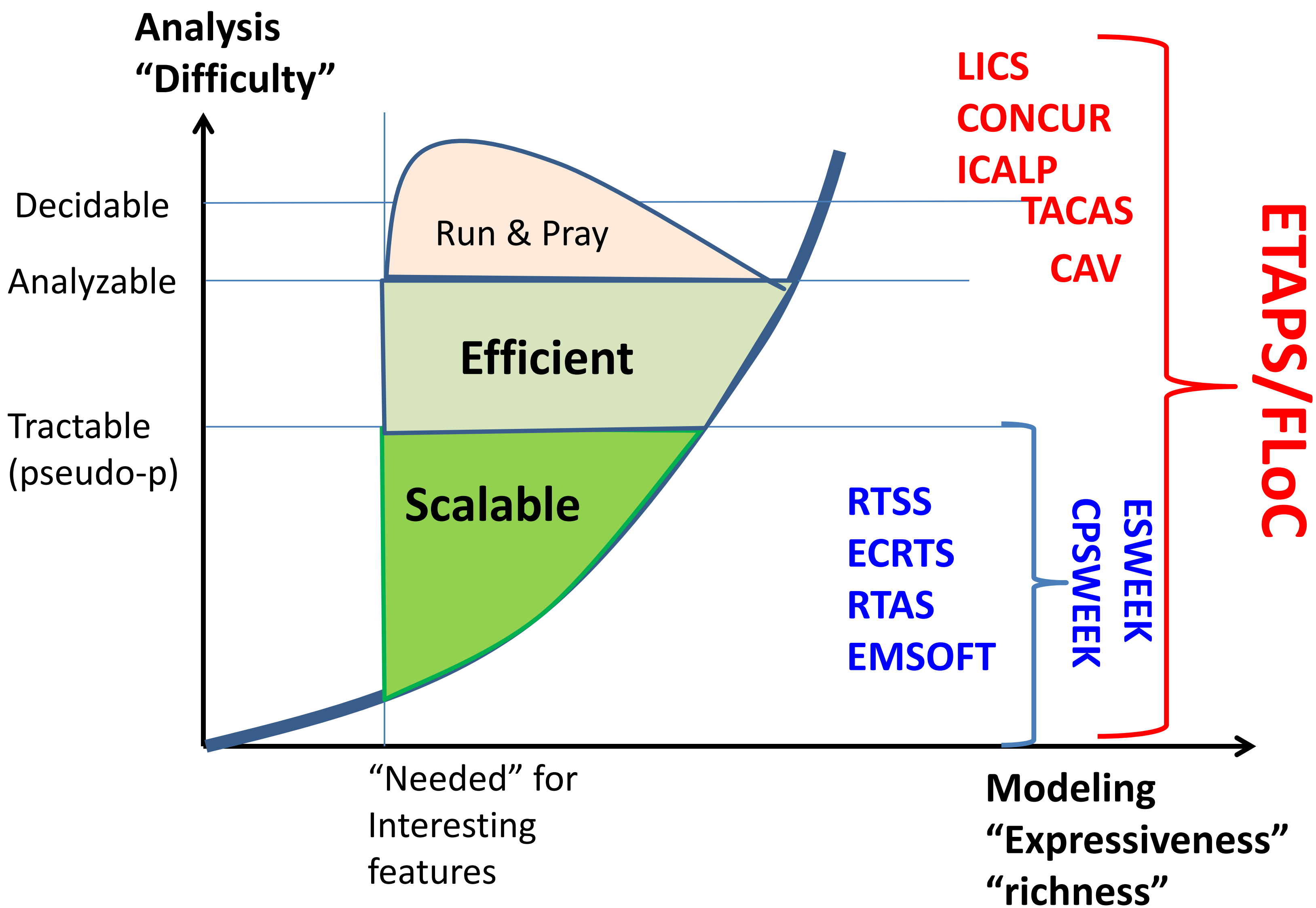
State of the art





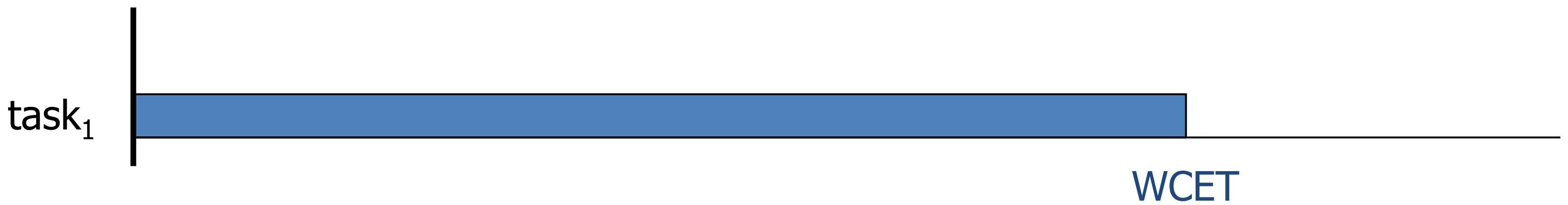
I can't solve the problem, neither can all these famous Model-Checkers

The Analyzable Zone of "Models"

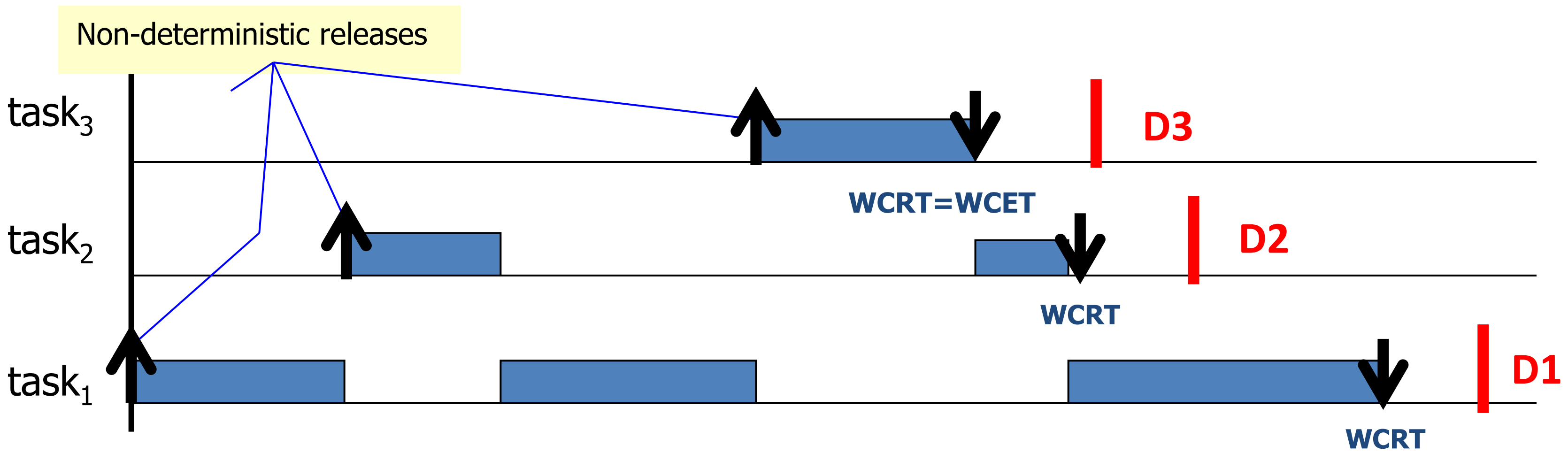


Timing Analysis

Sequential Case (WCET Analysis)



Concurrent Case (Response Time Analysis)



Timing Analysis

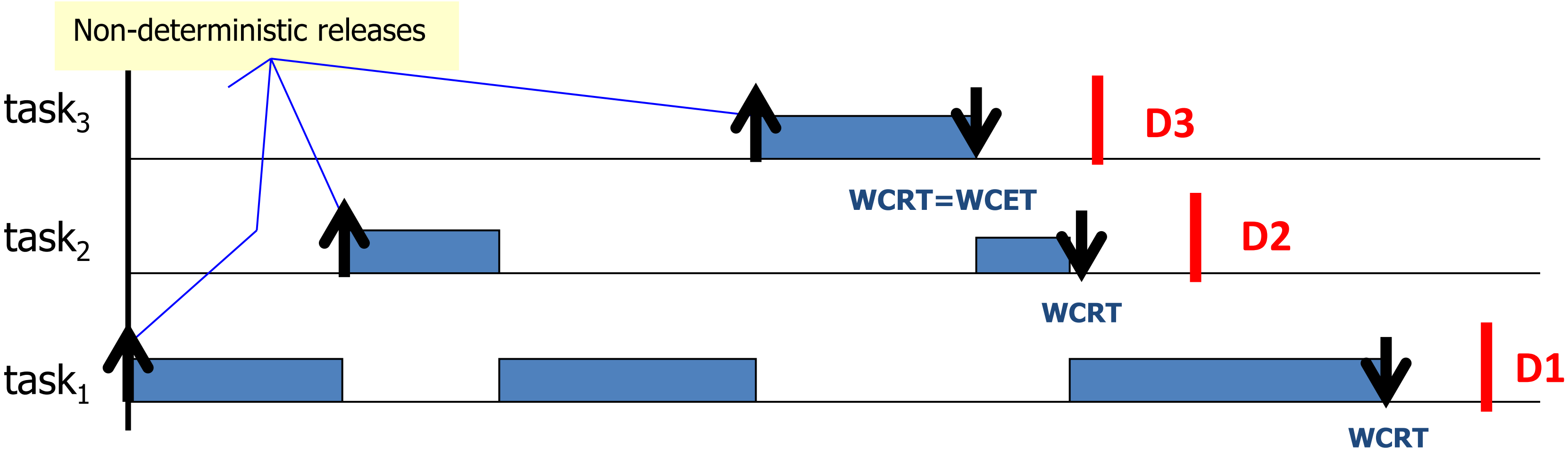
[aiT tool from AbsInt]

Wilhelm et al
Precision >> 95%

Sequential Case (WCET Analysis)

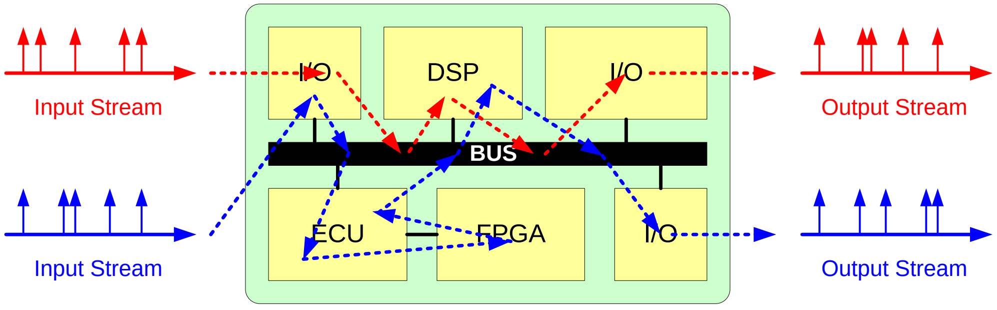
- Assume the WCET of each task is given (resource budget)
- How to estimate the Worst-Case Response Time of a task?

Concurrent Case (Response Time Analysis)



Modeling for (System-Level) Timing Analysis

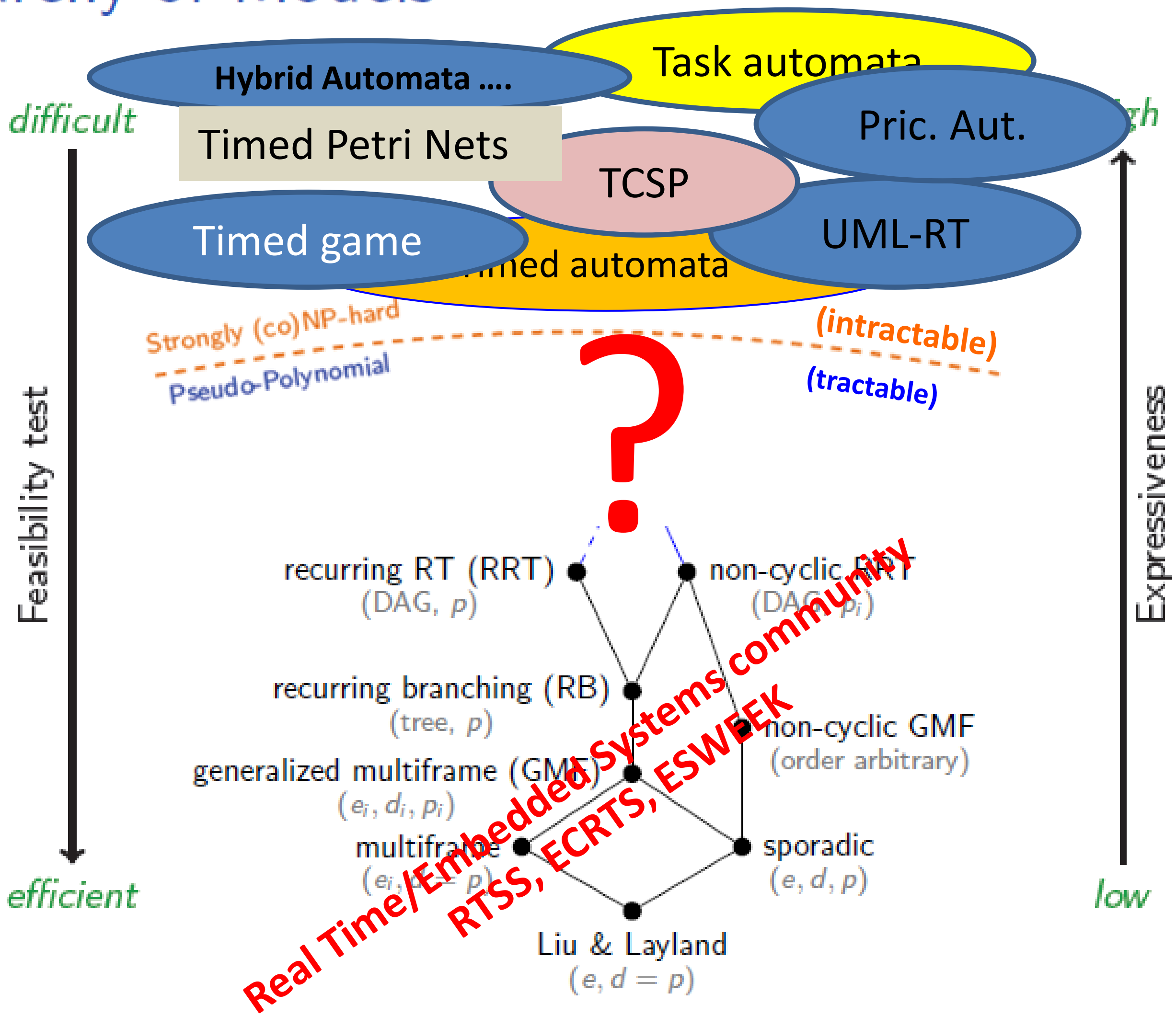
- The event arrival patterns e.g. using timed automata
- Synchronization between components,
- Resource arbitration, protocols and scheduling algorithms
- The resource demands or budget e.g. the WCET
- The timing constraints e.g. deadlines



Timed Models

- Timed Petri Nets, early 80s
  - Time Intervals over transition firing
- Process Algebras, 80s – 90s
  - Delays + untimed models e.g. Milner's CCS
- Timed Automata, early 90s
  - finite automata + clock constraints
- Real-Time Task Models since 70s
  - Layland and Liu's periodic tasks, 1973
  - The variants of L&L model [RTSS community]
- Real-Time Programming e.g. Ada 83
  - Delay, Tasking, Run-Time System
- Hybrid Systems/Automata, Modelica … UML RT … (yesterday)

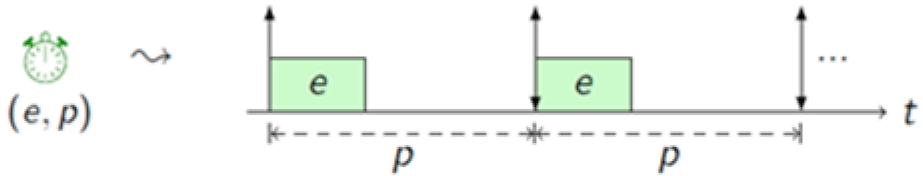
Hierarchy of Models



Liu and Layland's Model, 1973

A system is a set of periodic tasks each described by two numbers:

- **e**: the worst case execution time (WCET)
- **P**: the minimum inter-release delay (implicit deadline)



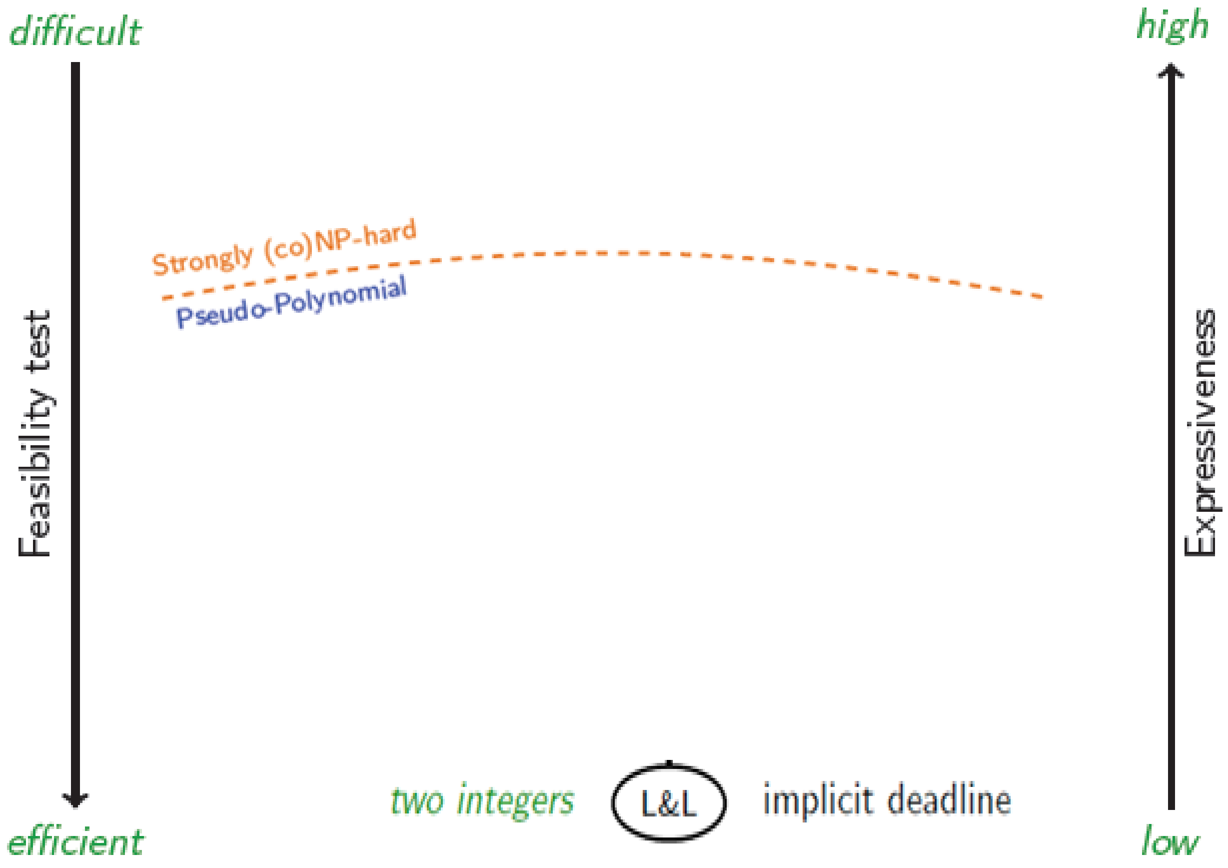
- **The workload of each task: e/p**
- **The system workload or utilization: U = ∑ ei/pi**

Feasibility (i.e. EDF-schedulability): no deadline miss if U ≤ 1

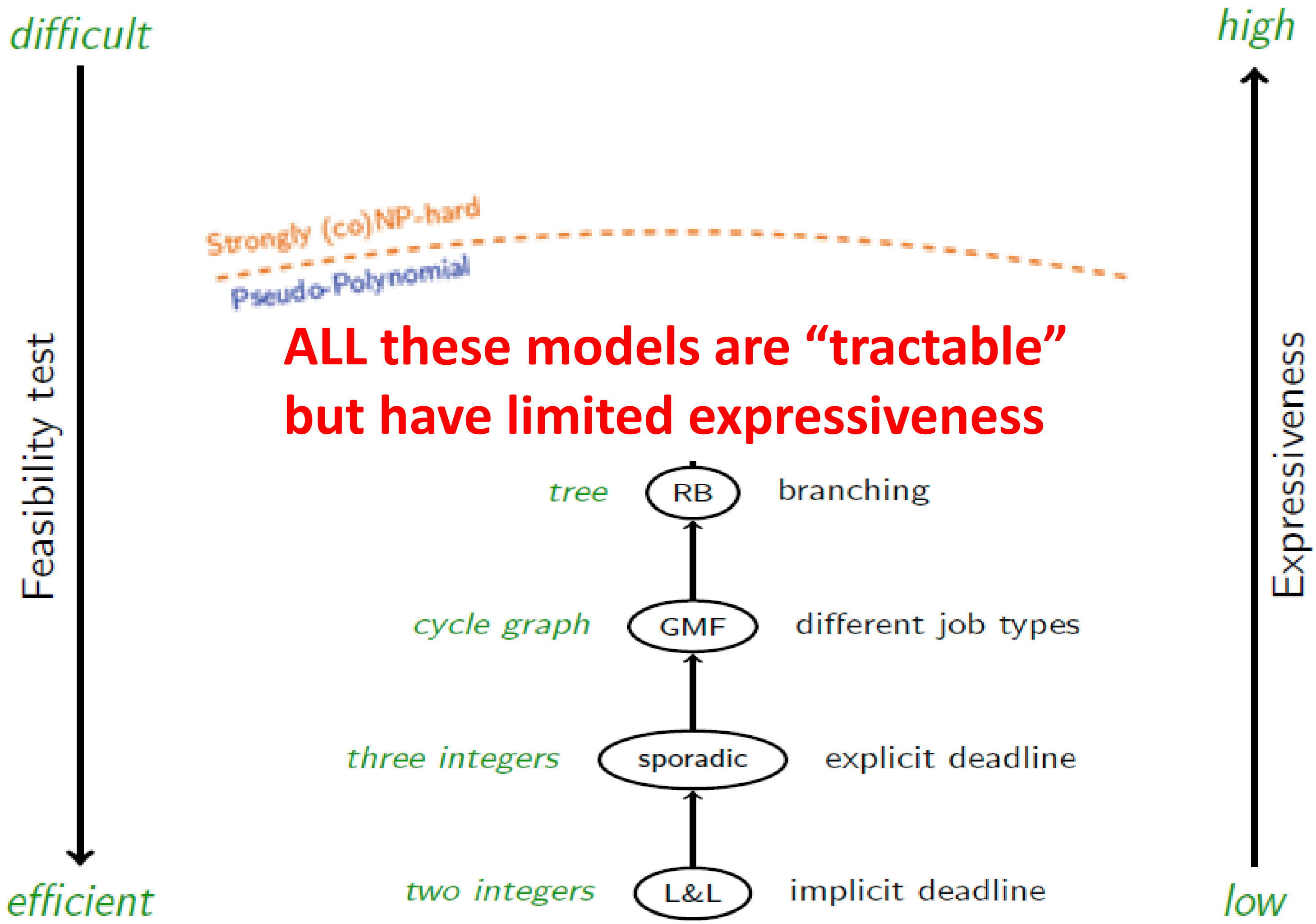
Fixed-priority Schedulability: no deadline miss if U ≤ $n(2^{1/n} - 1)$

The well-known Rate-Monotonic Scheduling

Hierarchy of Models



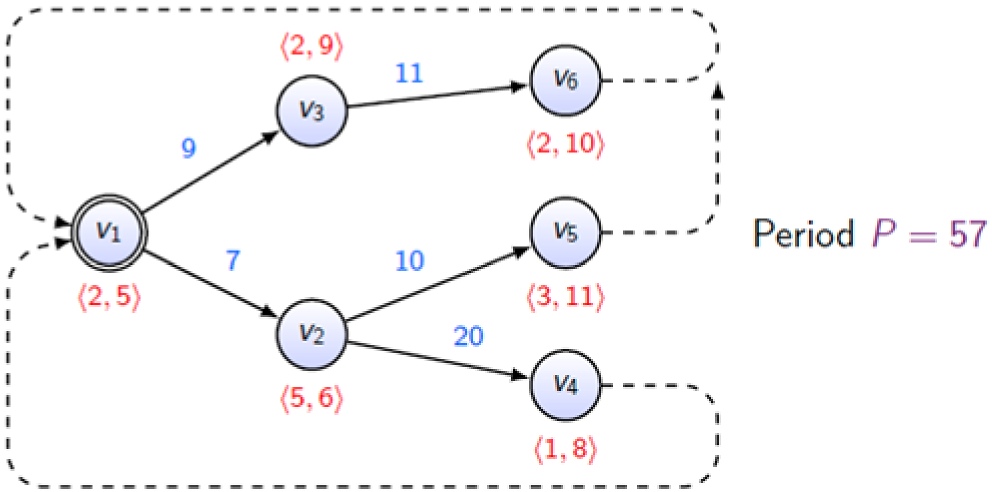
Hierarchy of Models

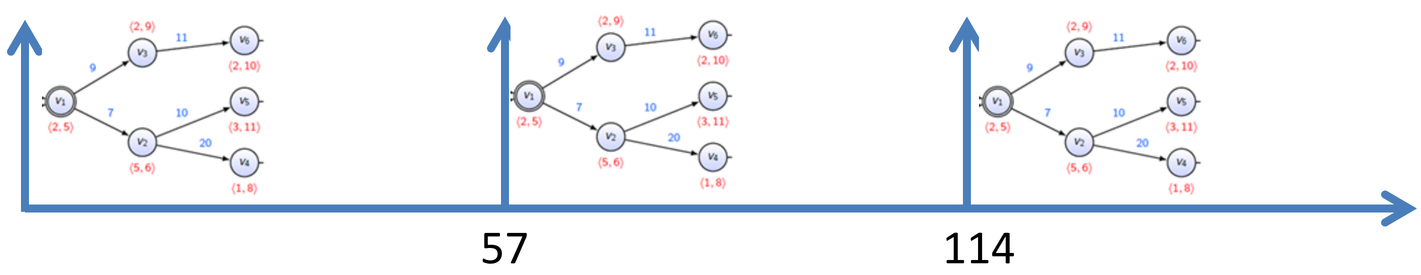


[Survey, RTS journal, Martin and Wang, 2015]

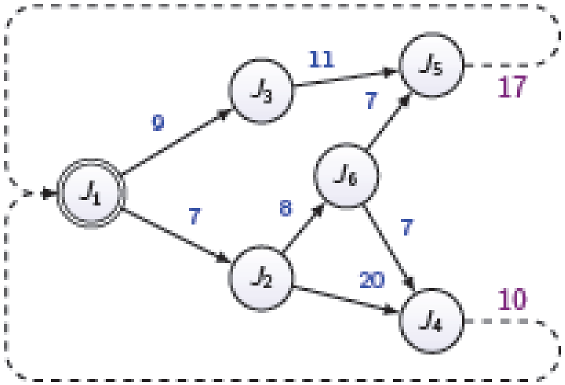
Example: Tree/DAG-task model

[Baruah et al, 1998, 2003, 2010]



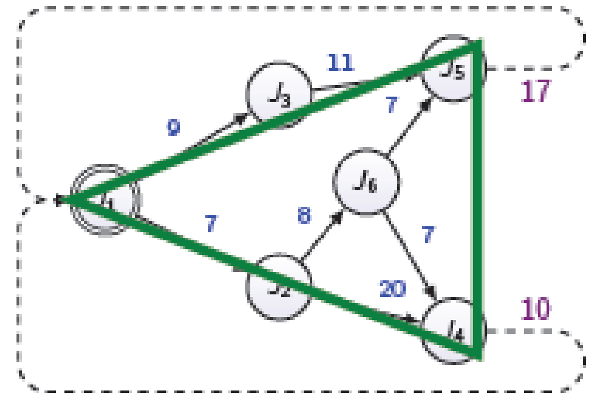


Restrictions of Non-Cyclic RRT

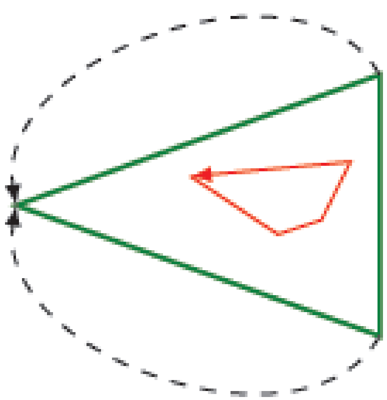


- Tasks are still *recurrent*
  - Always revisit source J_1
  - *No cycles allowed!*
- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)

Restrictions of Tree/DAG model

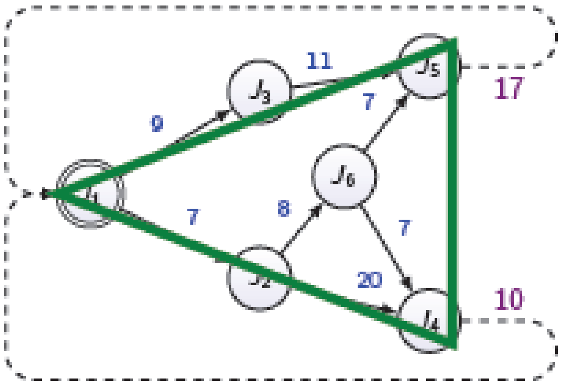


- Tasks are still *recurrent*
  - Always revisit source J_1
  - *No cycles allowed!*
- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)

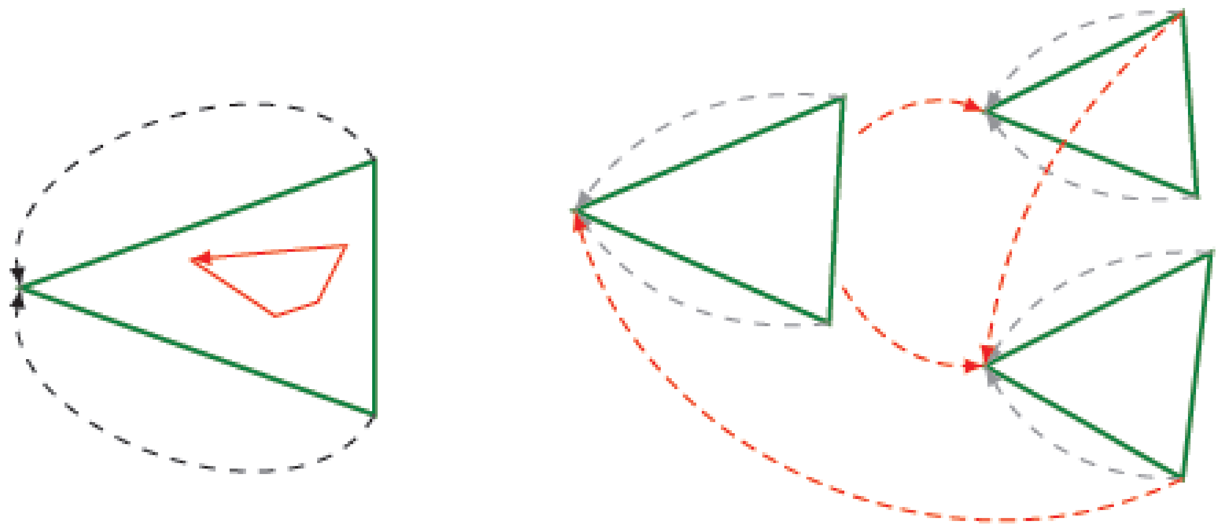


Restrictions of Tree/DAG model

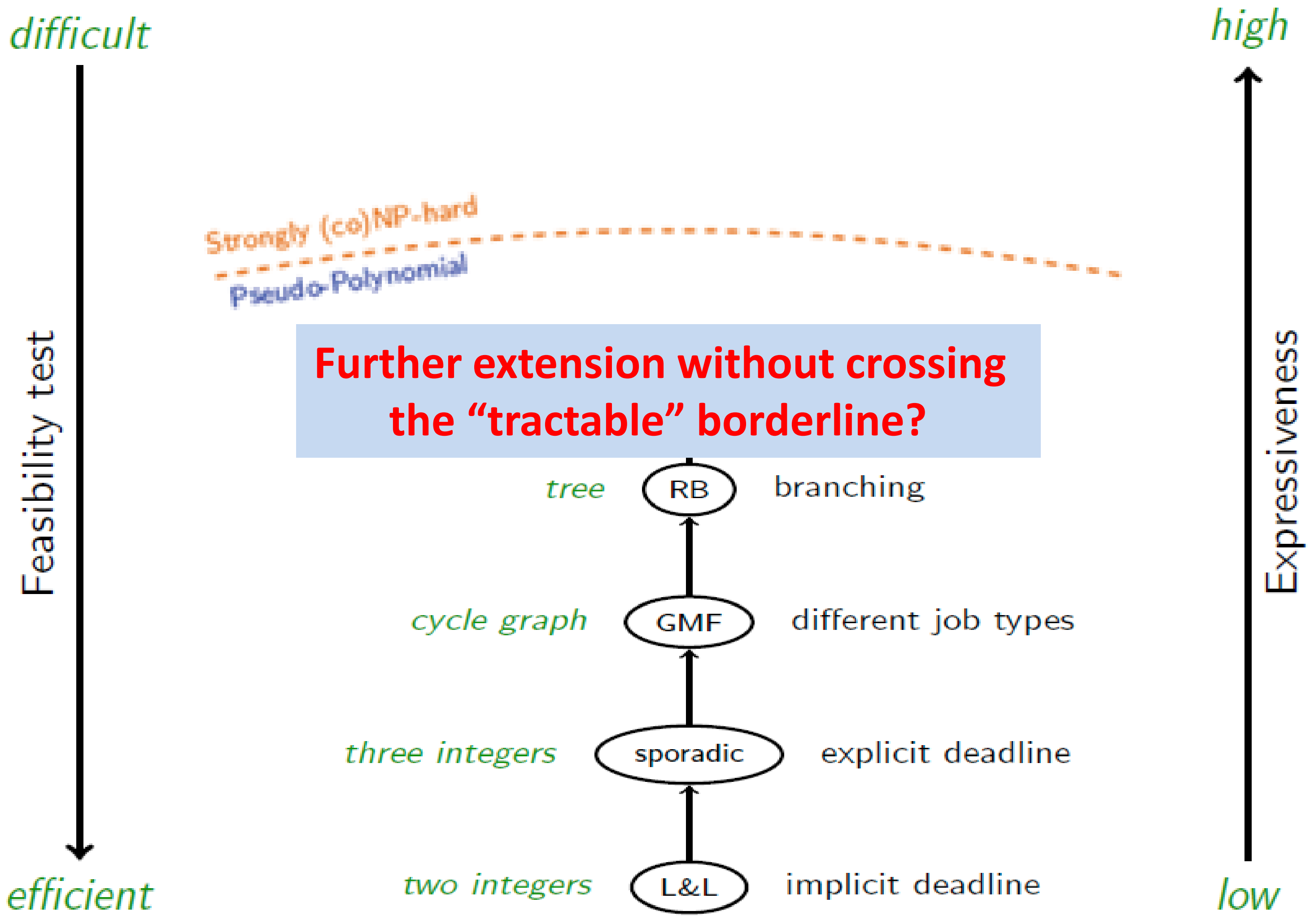
- Tasks are still *recurrent*
  - Always revisit source J_1
  - *No cycles allowed!*



- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)

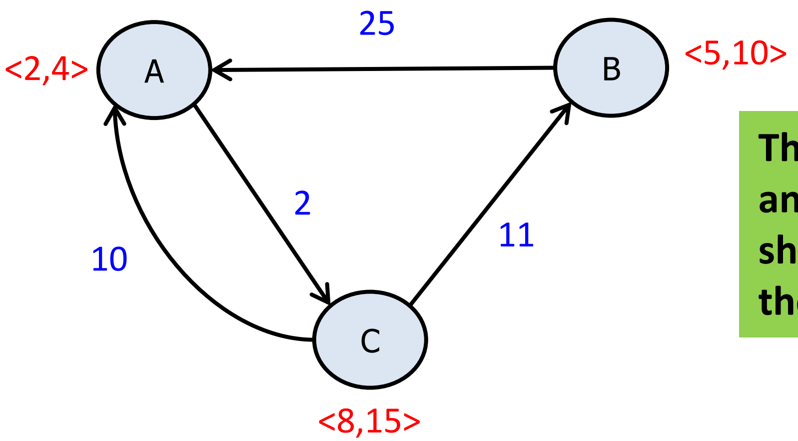


Hierarchy of Models



[Stigge et al, RTAS 2011]

The Digraph Real-Time Model (DRT)



The WCET, deadlines and release delays should be ensured by the Ada run-time system

- Pairs on nodes are the WCET and deadline on the task code
  e.g. A has WCET 2 and relative deadline 4
- Numbers on edges are the minimum inter-release delays

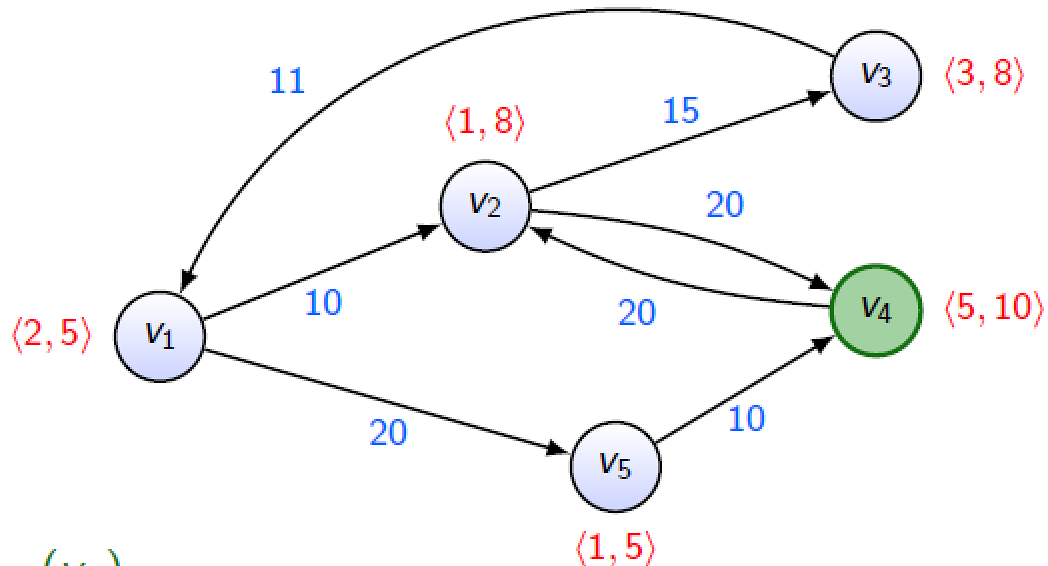
In Ada Tasking:

Procedure PA
“release A”
Delay(2);
PC

Procedure PB
“release B”;
Delay(25);
PA

Procedure PC
“release C”
If “condition”
then Delay(10); PA
else Delay (11); PB

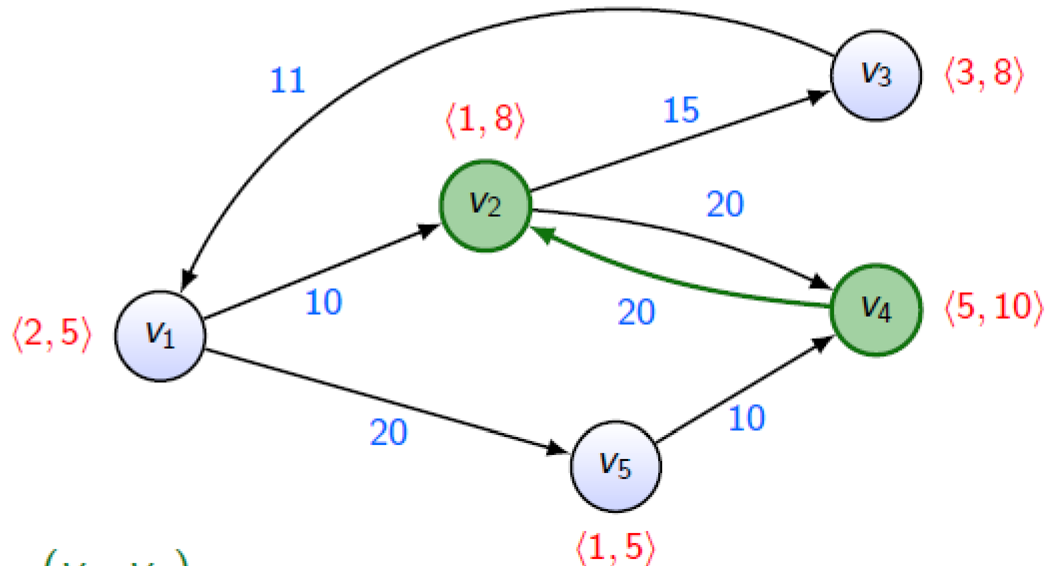
DRT: Semantics (any path of the graph is a possible behavior)



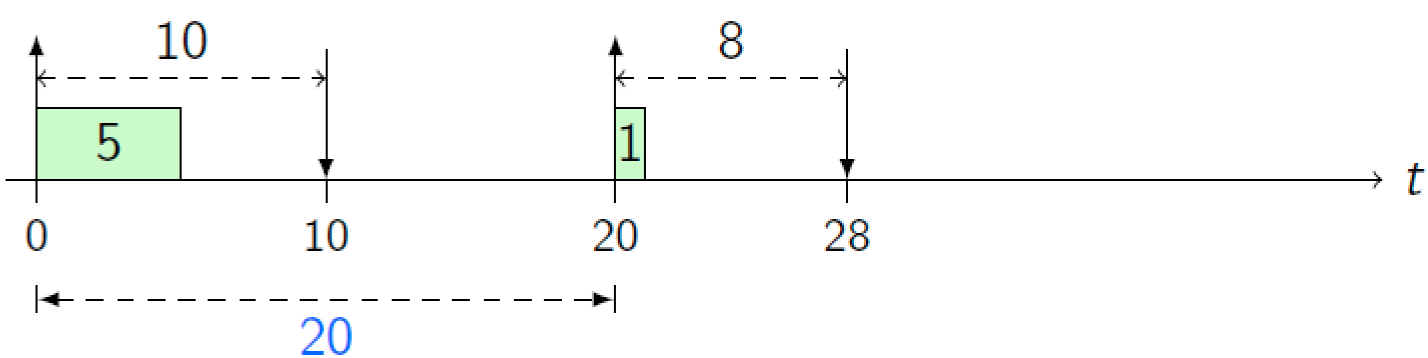
Path $\pi = (v_4)$

Demand bound: (10, 5)

DRT: Semantics (any path of the graph is a possible behavior)



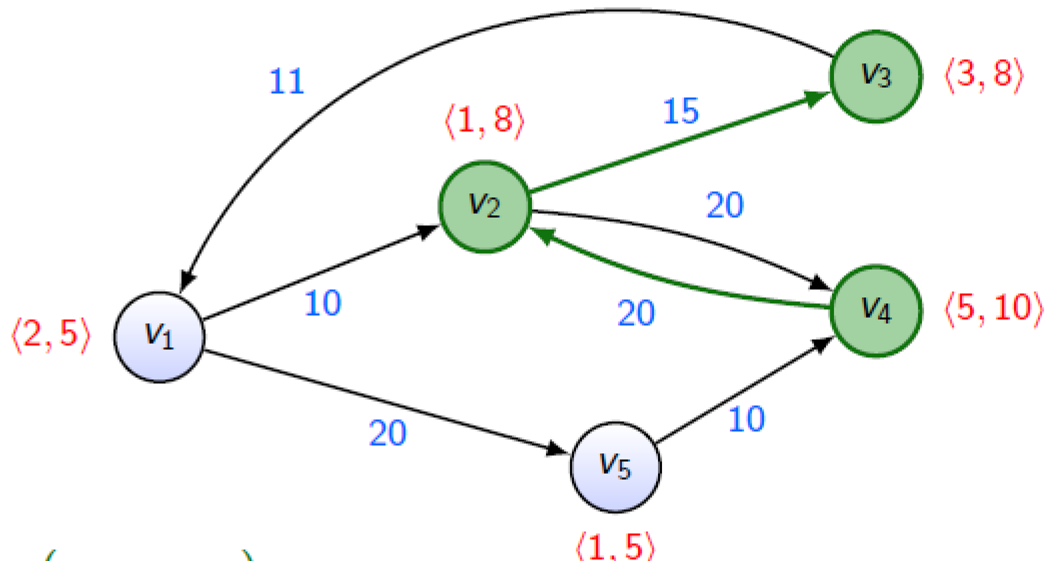
Path $\pi = (v_4, v_2)$



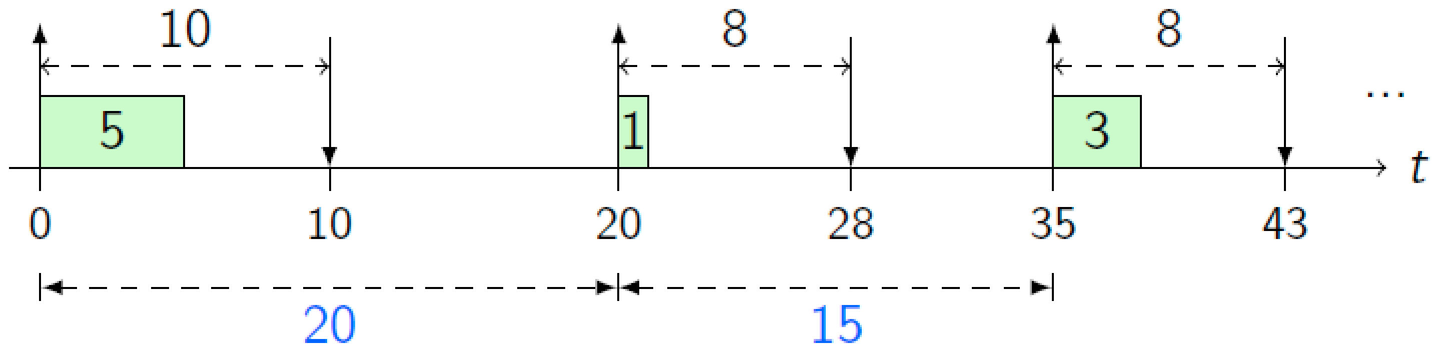
Demand bound : (10, 5)

Demand bound : (28, 6)

DRT: Semantics (any path of the graph is a possible behavior)

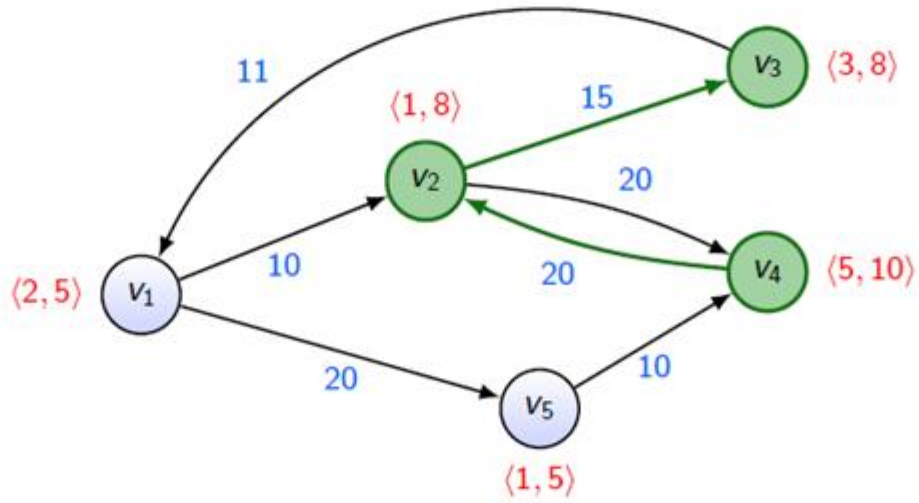


Path $\pi = (v_4, v_2, v_3)$

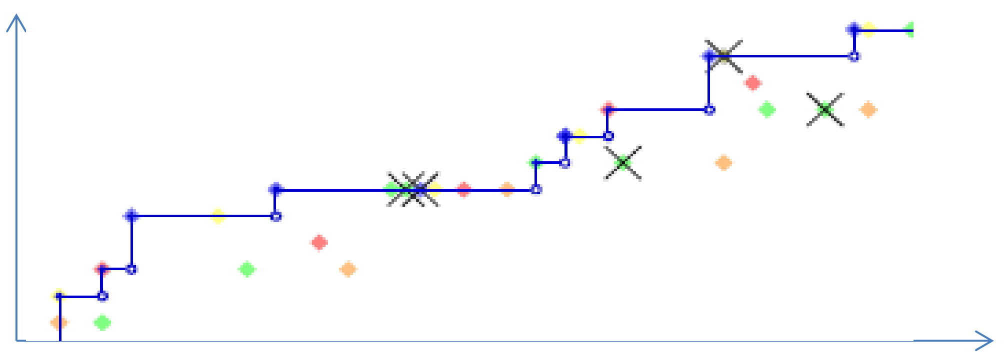


Workload: (10, 5) **Workload: (28, 6)** **Demand bound : (43, 9)**

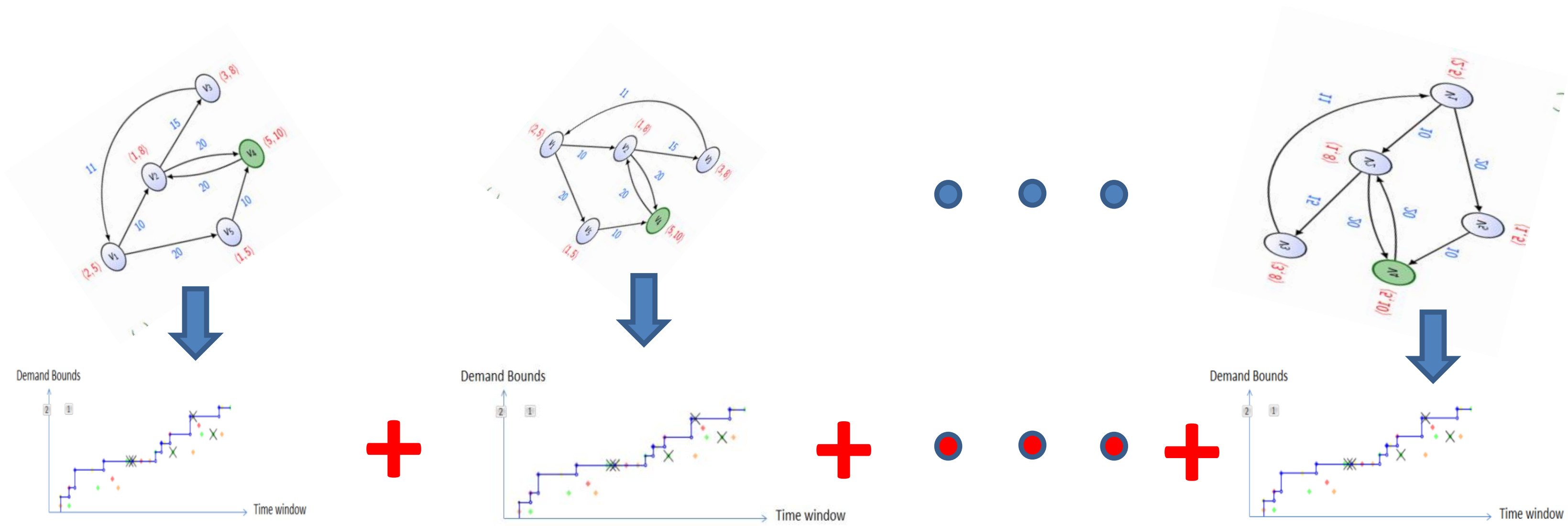
Workload of a DRT



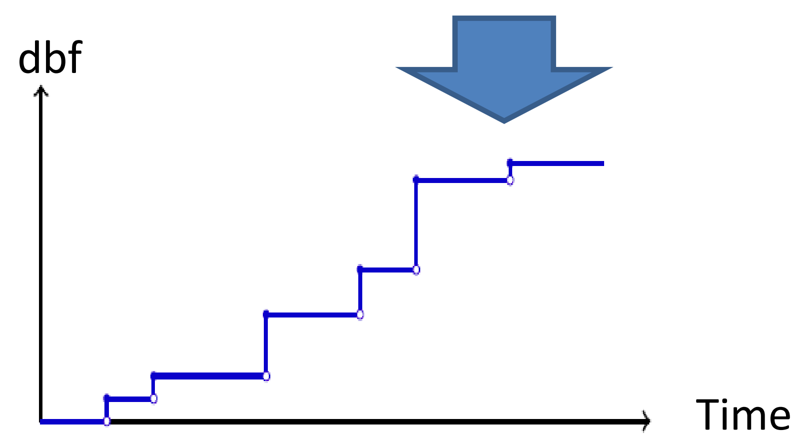
Demand Bounds Function (dbf)



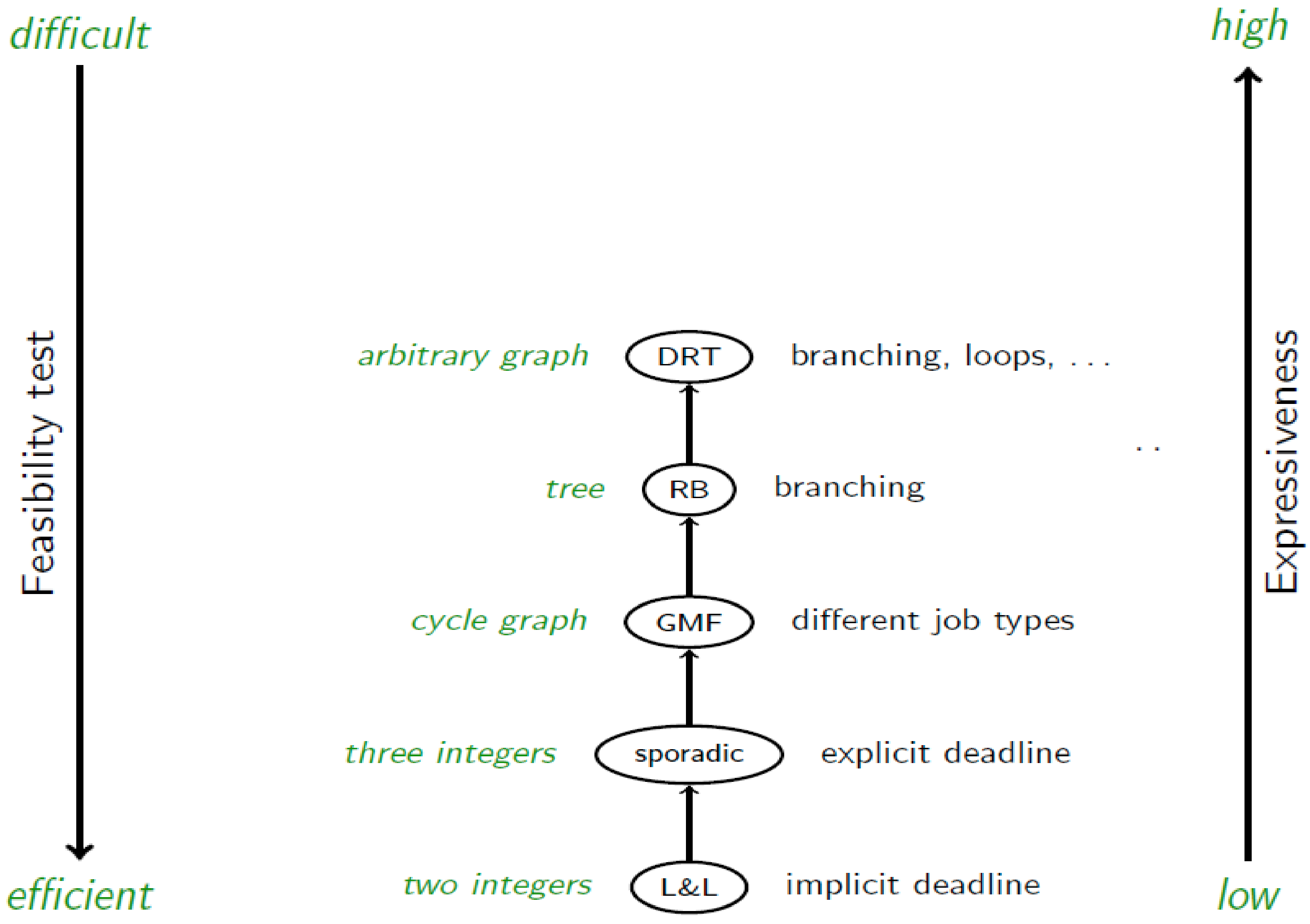
A **system model** = a set of DRT’s modeling the components



The system workload:

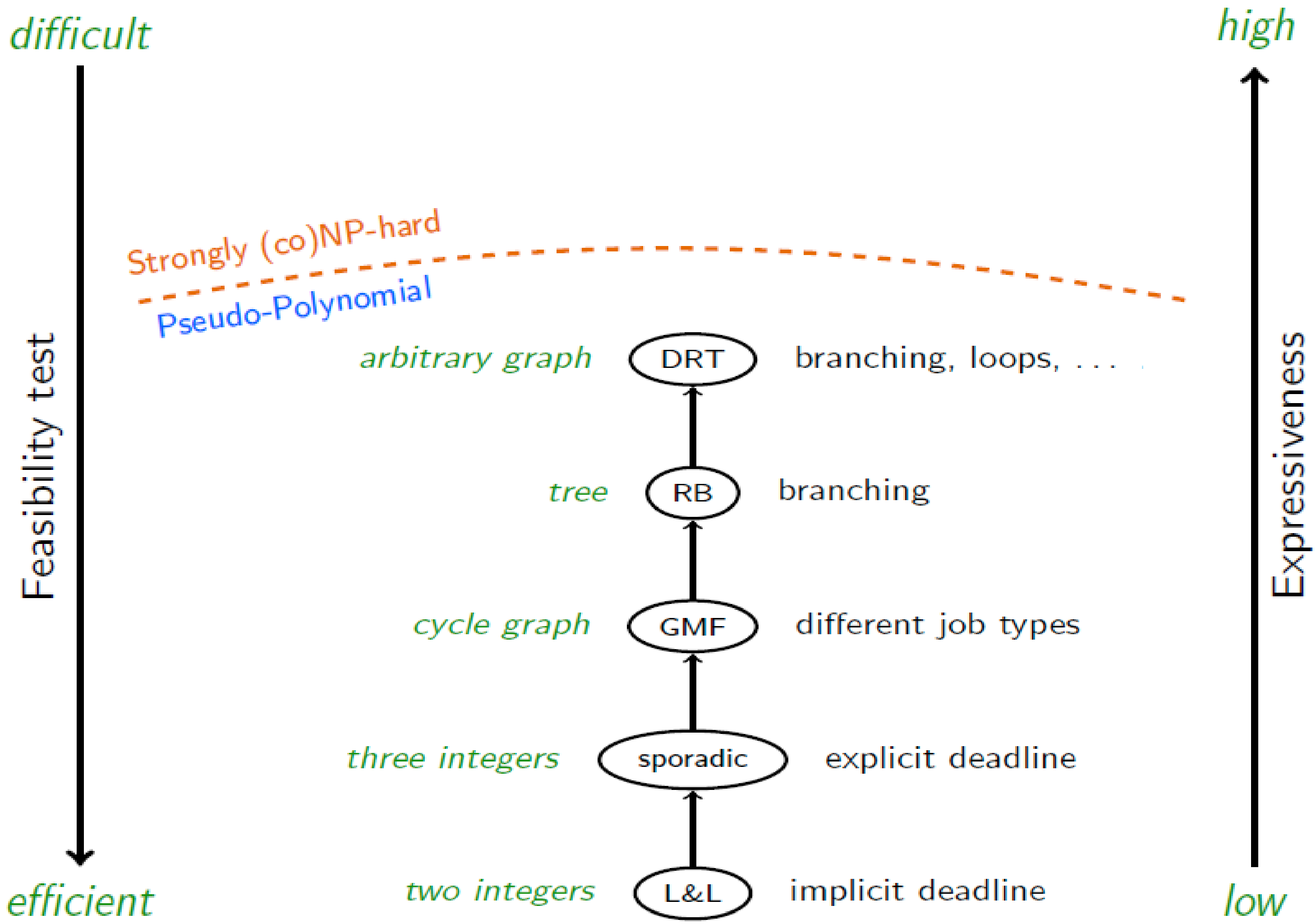


Hierarchy of Models



Hierarchy of Models

[Stigge et al, RTAS 2011]



Complexity Result **[RTAS 2011]**

Theorem (S. et al., 2011)

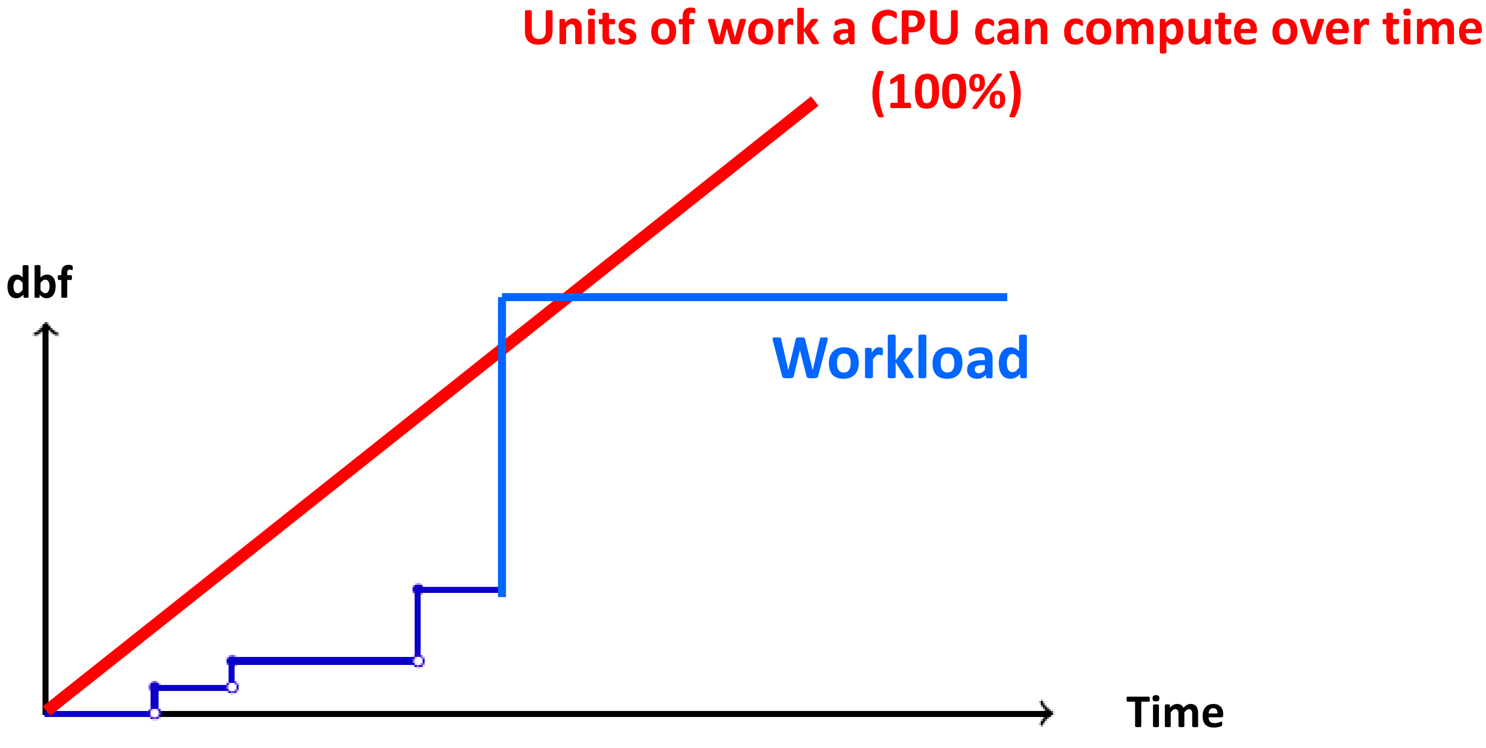
For DRT task systems τ with a utilization bounded by any $c < 1$, feasibility can be decided in pseudo-polynomial time.

Pseudo-polynomial time = Tractable/efficient

M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph Real-Time Task Model," in *Proc. of RTAS 2011*, pp. 71–80.

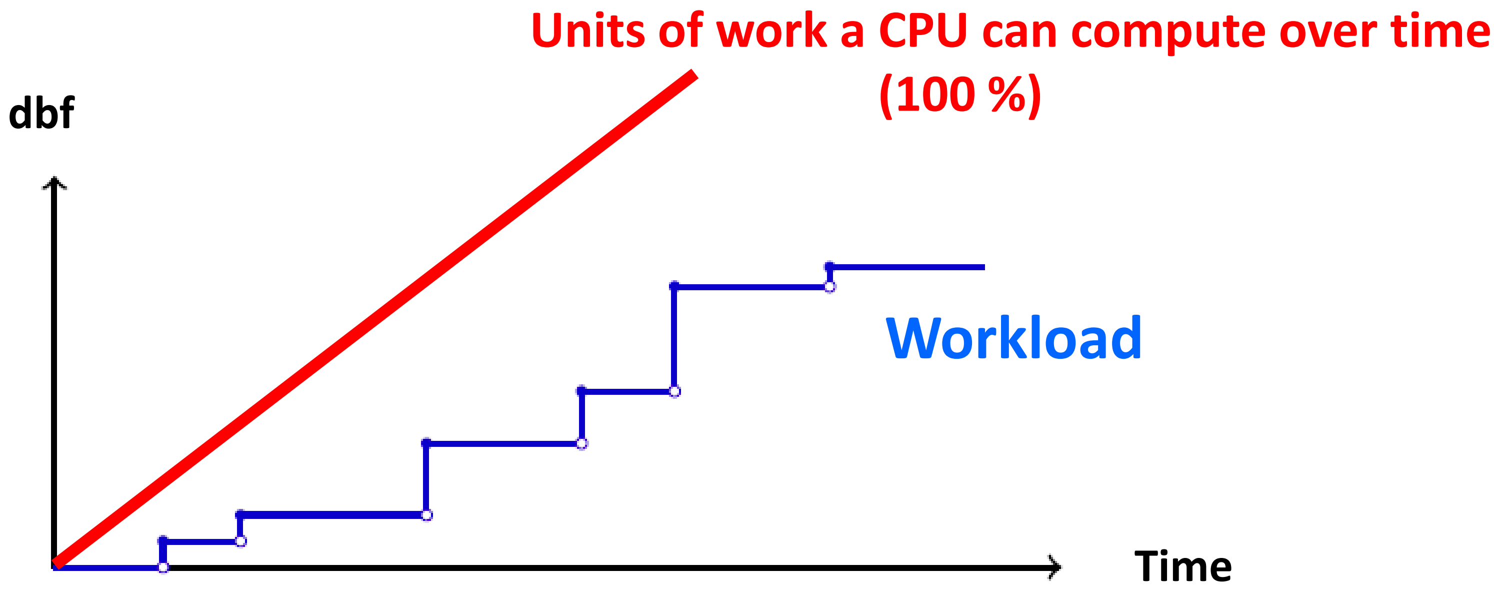
Ideas for feasibility analysis

- Characterize the system workload …
- If the worst-case workload is over 100%, it is over-loaded, implying deadline miss



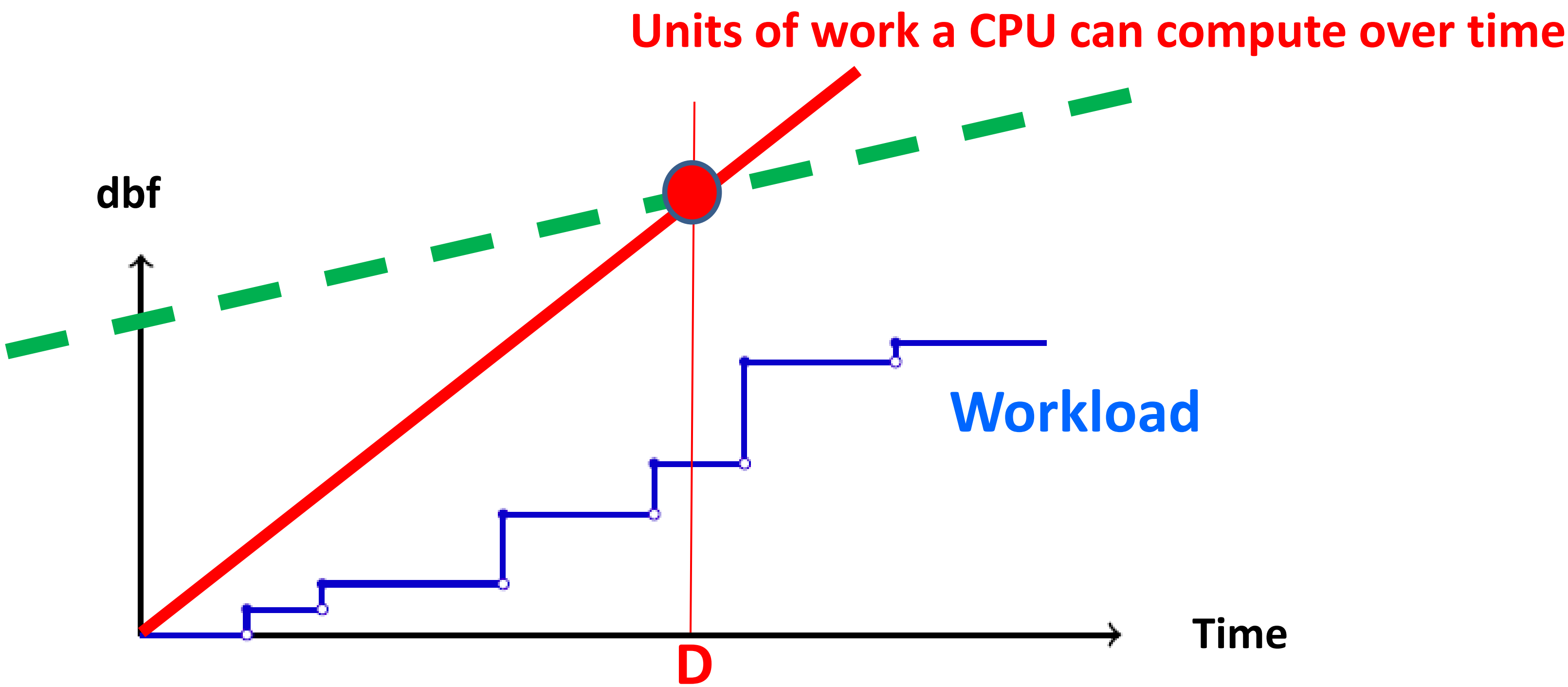
How to check this?

Of course, if the **BLUE line** is always below the **RED**, the system should work well without deadline miss!

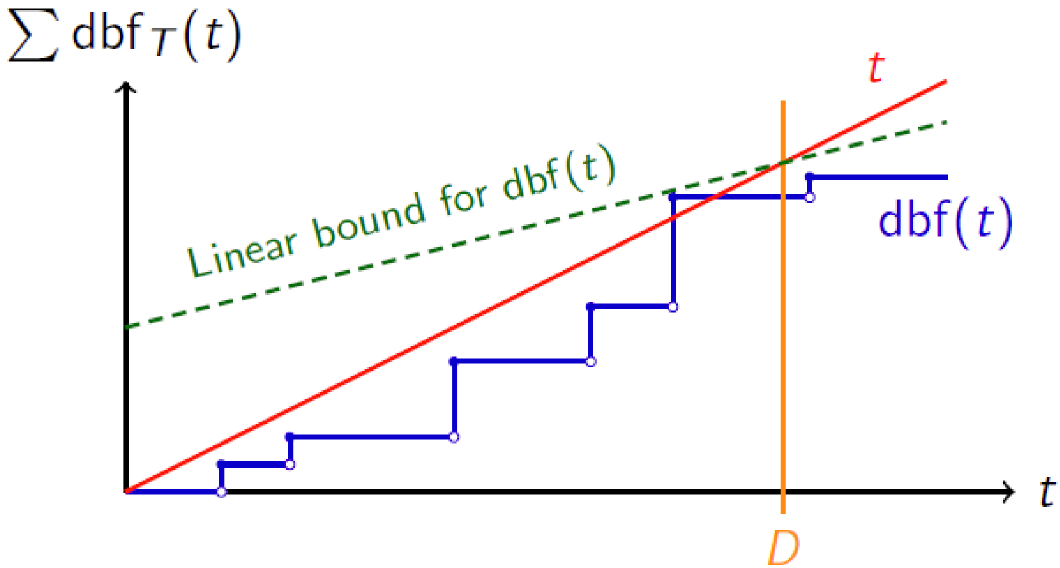


Here is the intuition why “Pseudo-P”

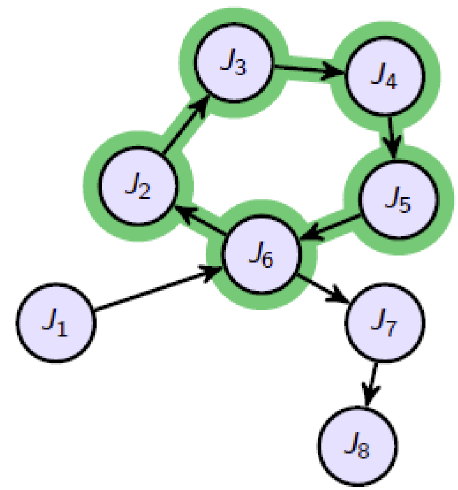
If the **utilization (long-term rates of DRT’s)** of a system is bounded by a constant **c < 1,** any deadline miss, if exists, must appear before a **pseudo-polynomial upper bound:**



Calculating the Bound



"Most dense" cycle

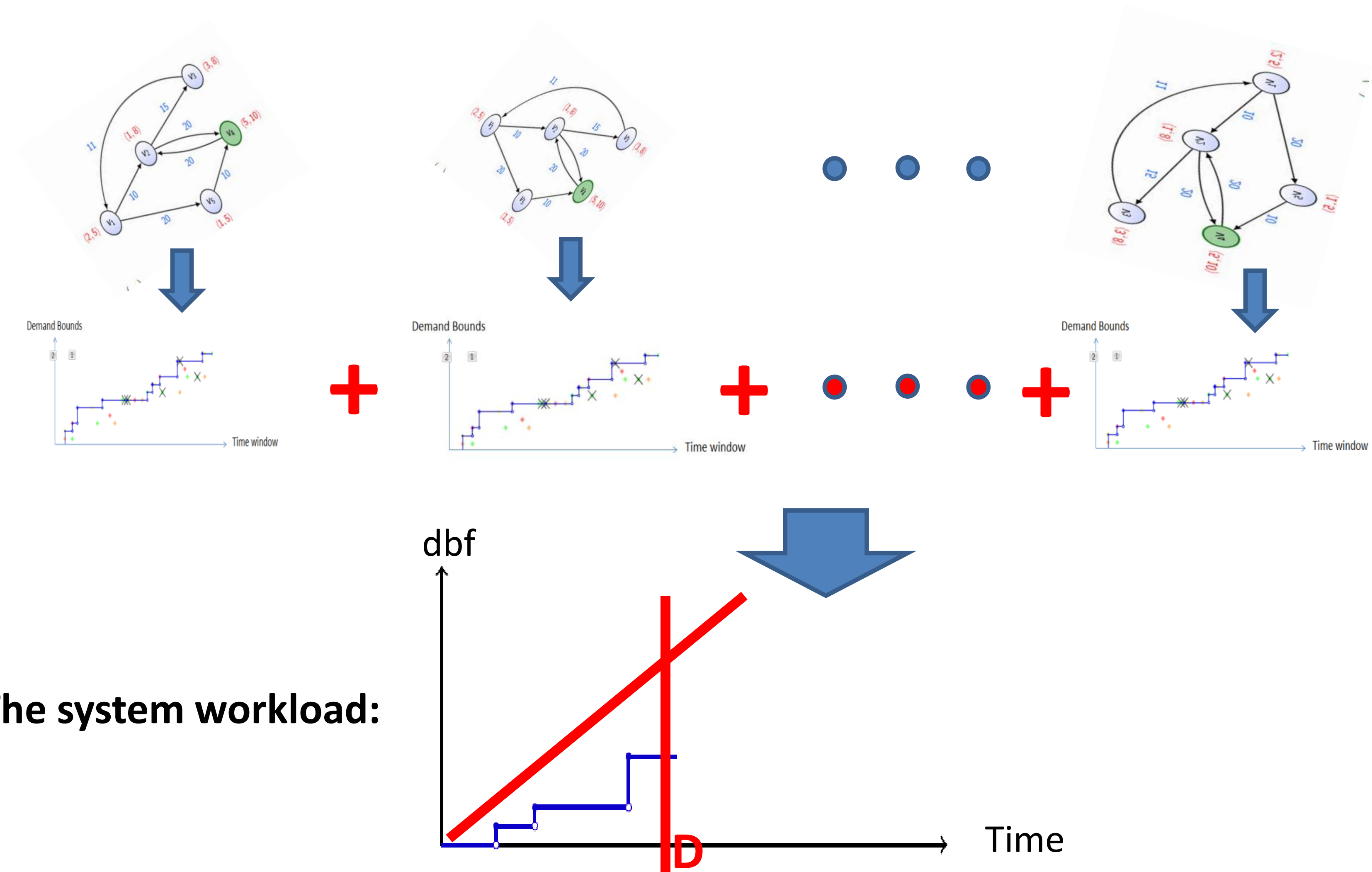


$$\mathrm{dbf}(t) \leqslant t \cdot U(\tau) + e^{sum}$$

- Linear bound for dbf(t)
  - Slope: Less than 1
- Intersection with t gives bound D
- Check only up to D

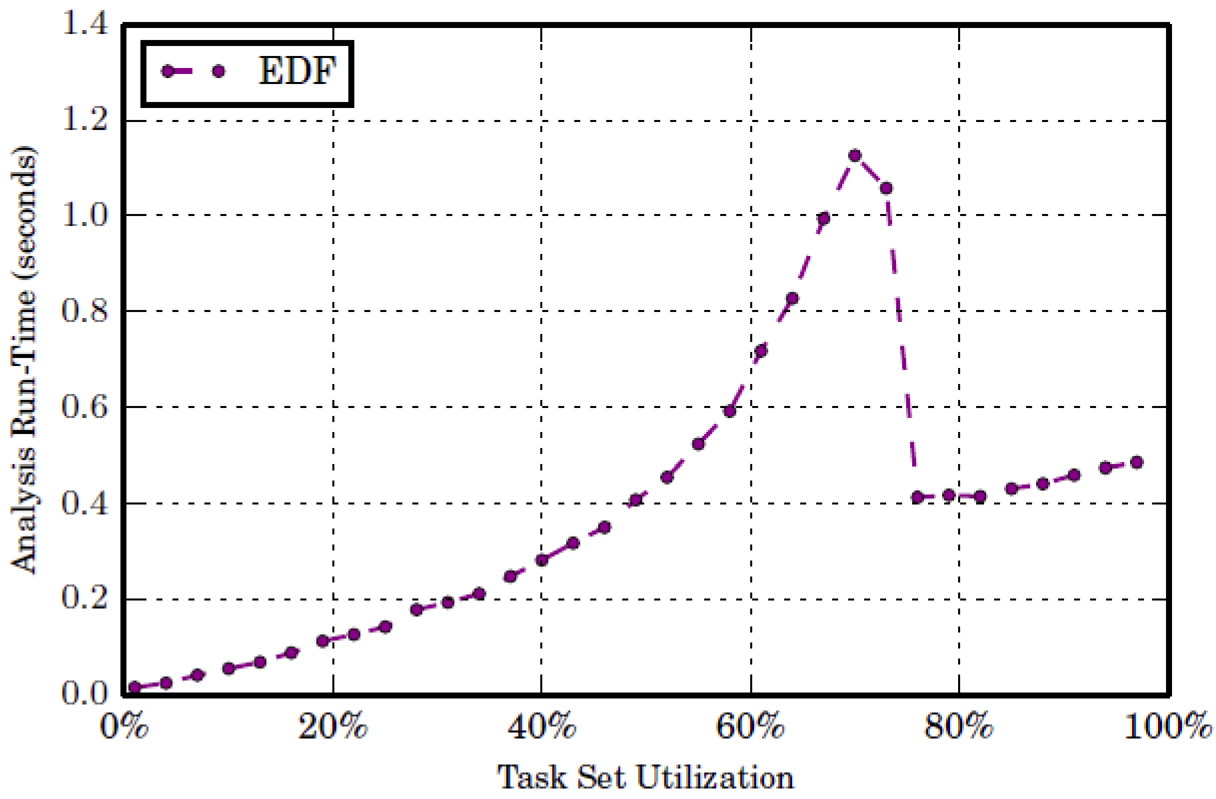
$$D = \frac{e^{sum}}{1 - U(\tau)}$$

A **system model** = a set of DRT's modeling the components



The system workload:

Evaluation: Runtime vs. Utilization



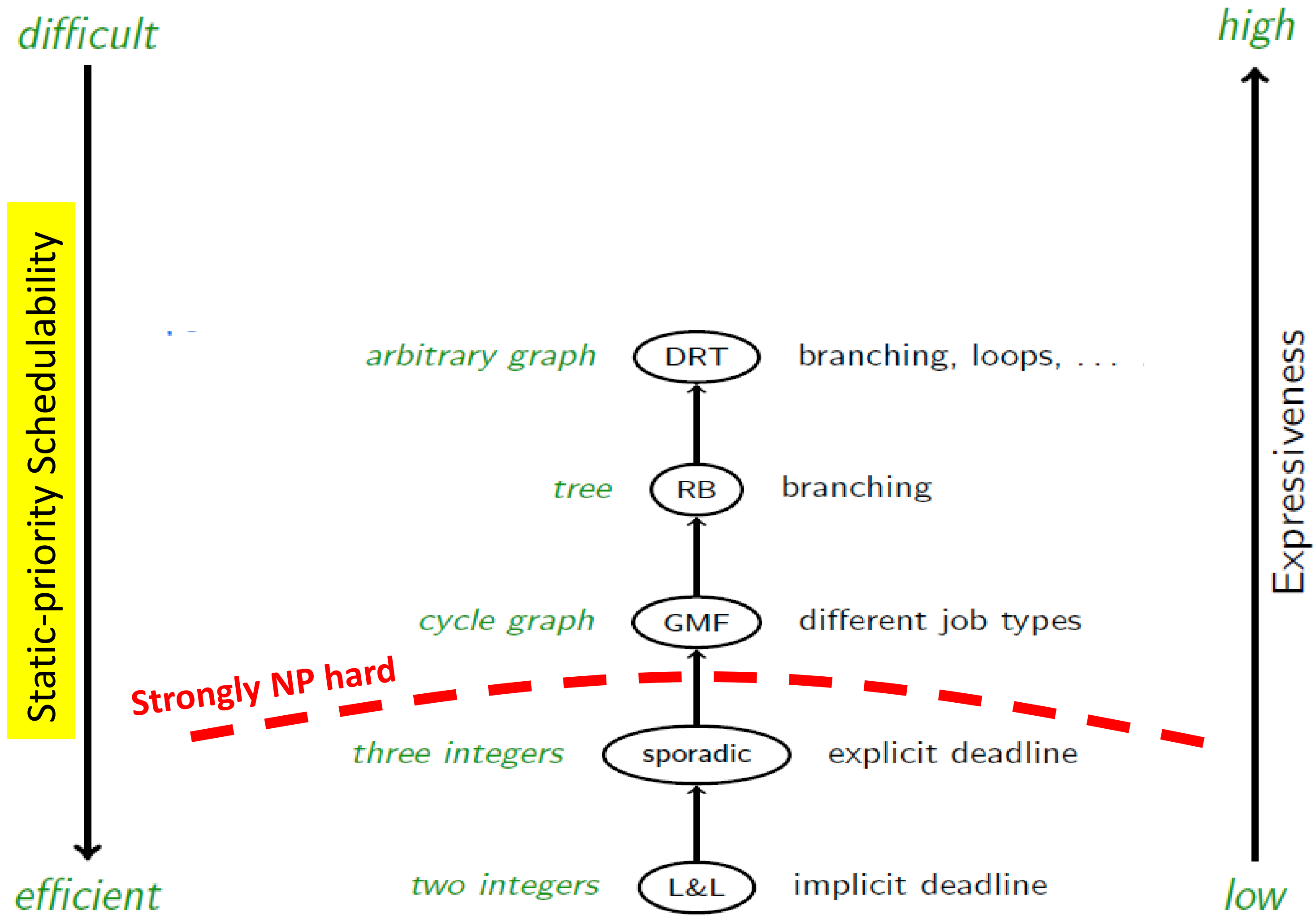
Setting:

- Randomly generated task sets
- 1-30 tasks, 5-10 vertices per task, branching degree 1-3, . . .

- How about synchronization?
  - the analysis without considering synchronization is SAFE!
  - Precise analysis possible with “Combinatorial Refinement”
- How about “static priority scheduling”?

Hierarchy of Models

[Stigge/Wang, ECRTS 2012]



Summary

Models	Analysis Complexity	
	Feasibility i.e. EDF-Schedulability	Static-priority Schedulability
General graphs (Di-graph)	Pseudo-P	Strongly coNP-complete
Trees/DAGs	Pseudo-P	Strongly coNP-complete
Cyclic graphs (GMF)	Pseudo-P	Strongly coNP-complete
Sporadic (L&L, deadline≠period)	Pseudo-P	Pseudo-P
L&L (periodic)	Linear	Pseudo-P

For systems with utilization bounded by a constant less than 1 (or below 100%)

Otherwise Strongly coNP-complete

[ECRTS 2012]

What can we do?

!! The problem open for 25 years, theoretically interesting !!

[ECRTS 2015, Pontus Ekberg and Wang Yi]

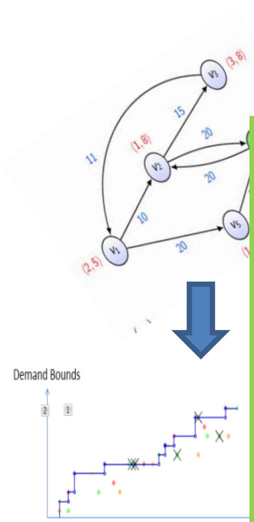
[TACAS 2015]

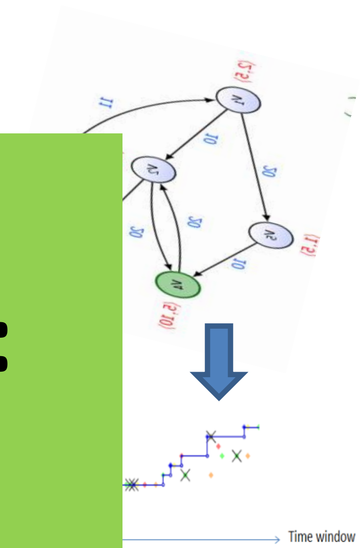
Combinatorial Refinement

solving "Combinatorial Problems"

(for timing analysis, it works very well!)

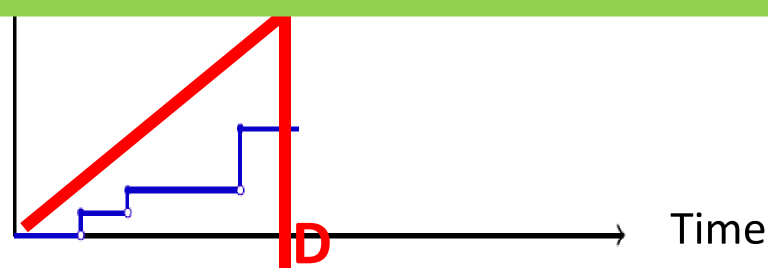
A **system model** = a set of DRT's modeling the components



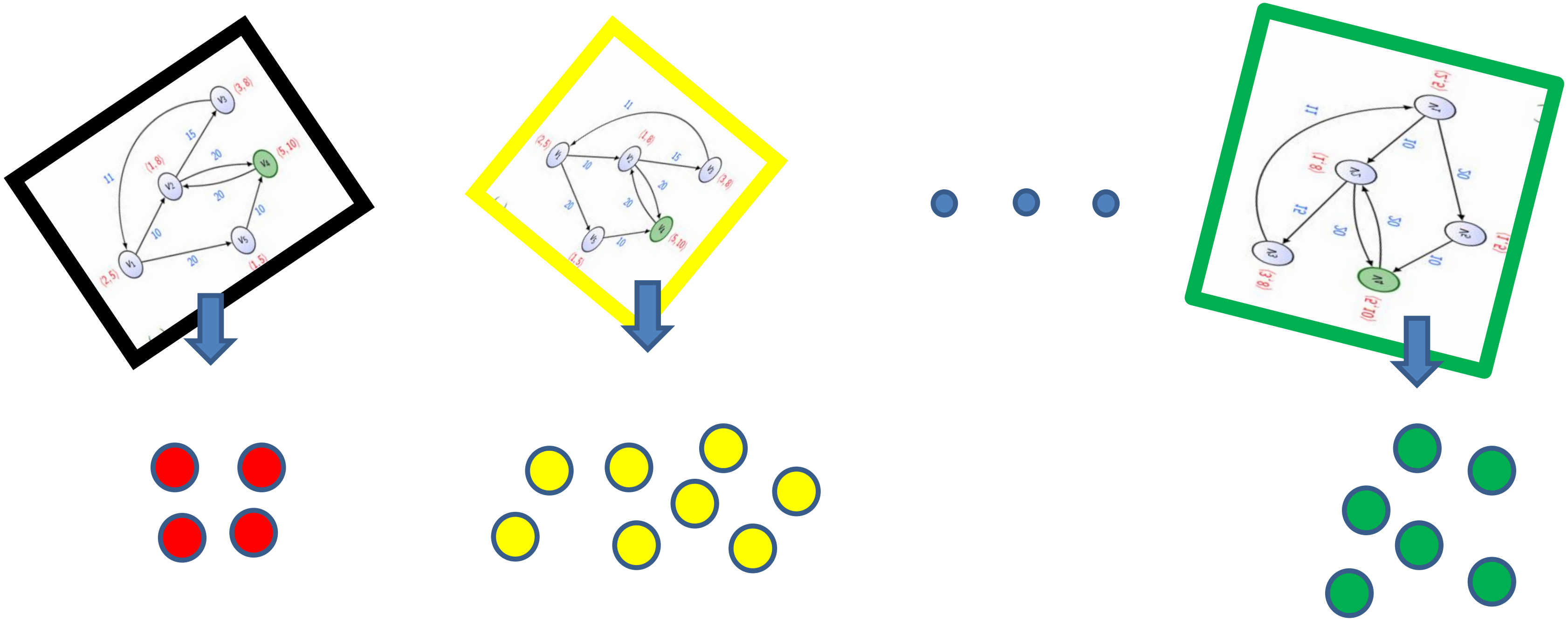


This works perfectly for feasibility checking: the global worst case can be constructed from the local worst cases

The system workload:

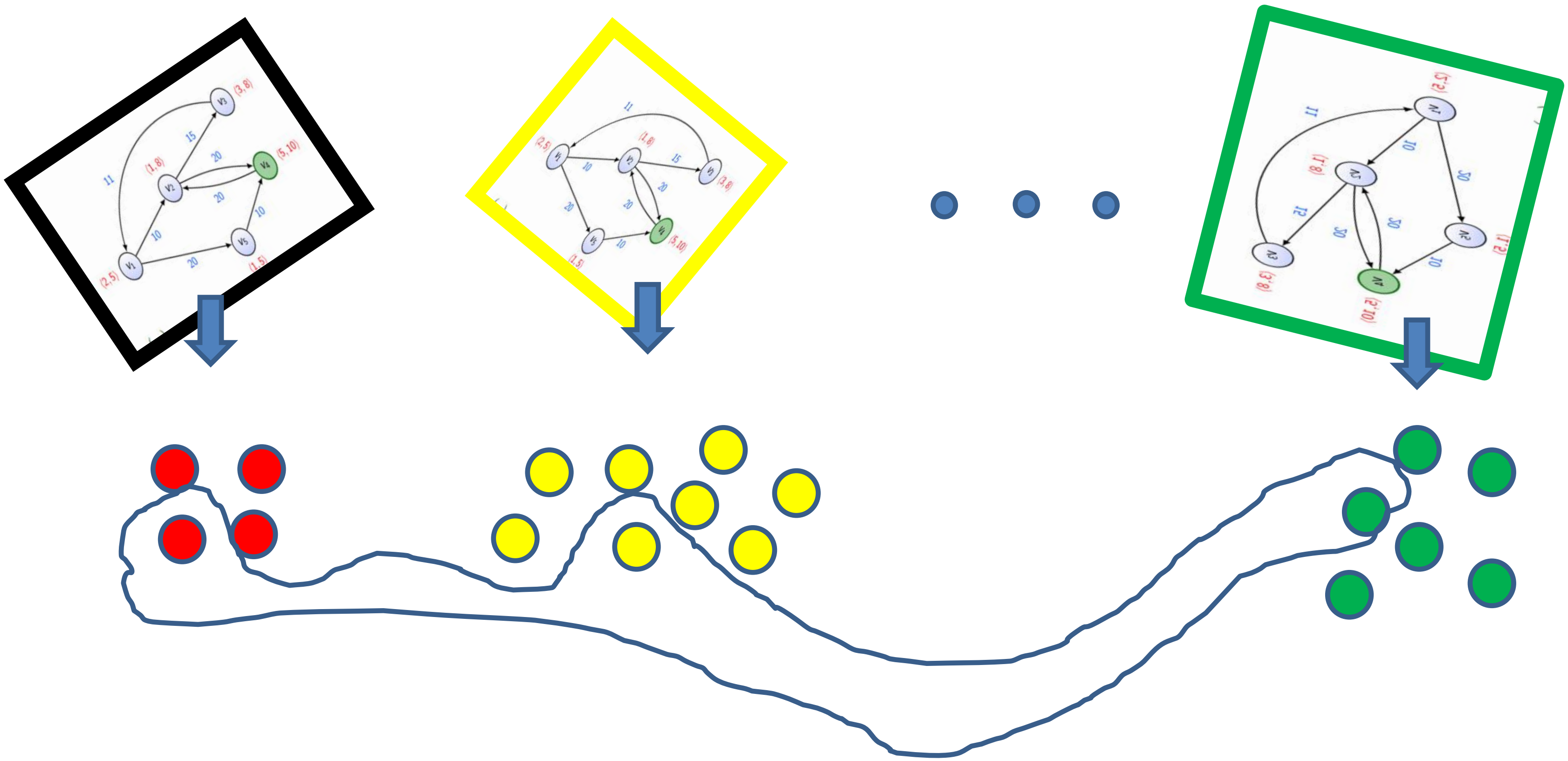


A **system model** = a set of DRT's modeling the components



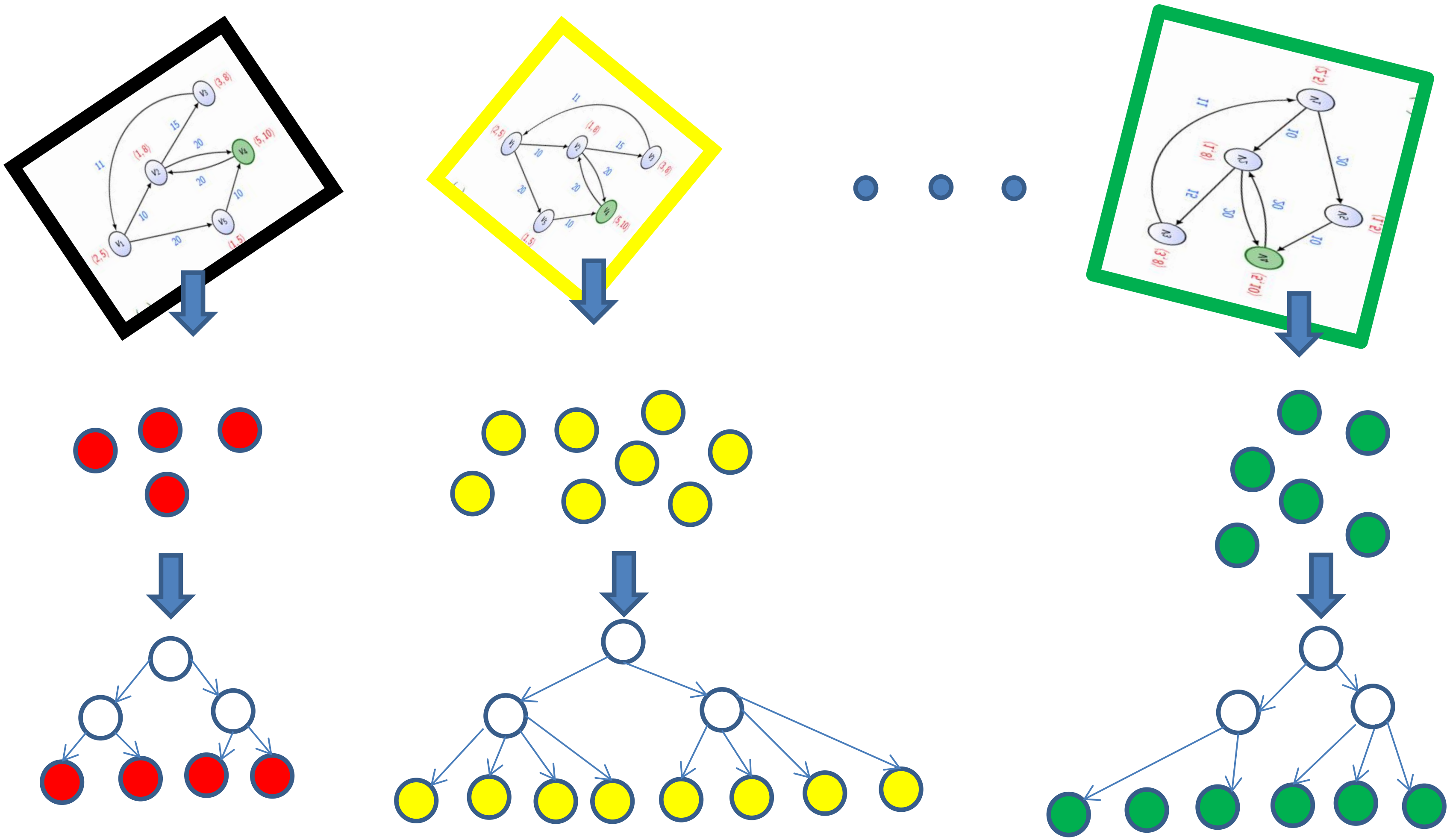
In general, each component may have a set of behaviors e.g. Paths or traces

A **system model** = a set of DRT's modeling the components

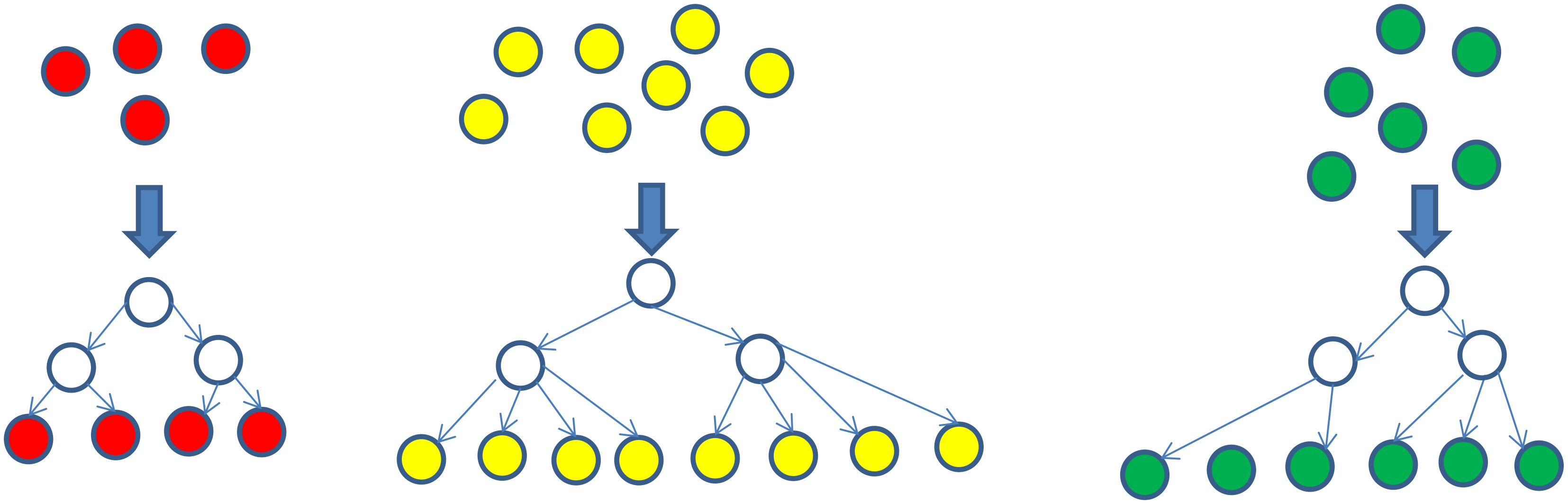


Often, we have to check some property guaranteed by all the combinations of individual local behaviors and thus may have to enumerate … (combinatorial explosion)

Construct an **Abstract Tree** for each individual component



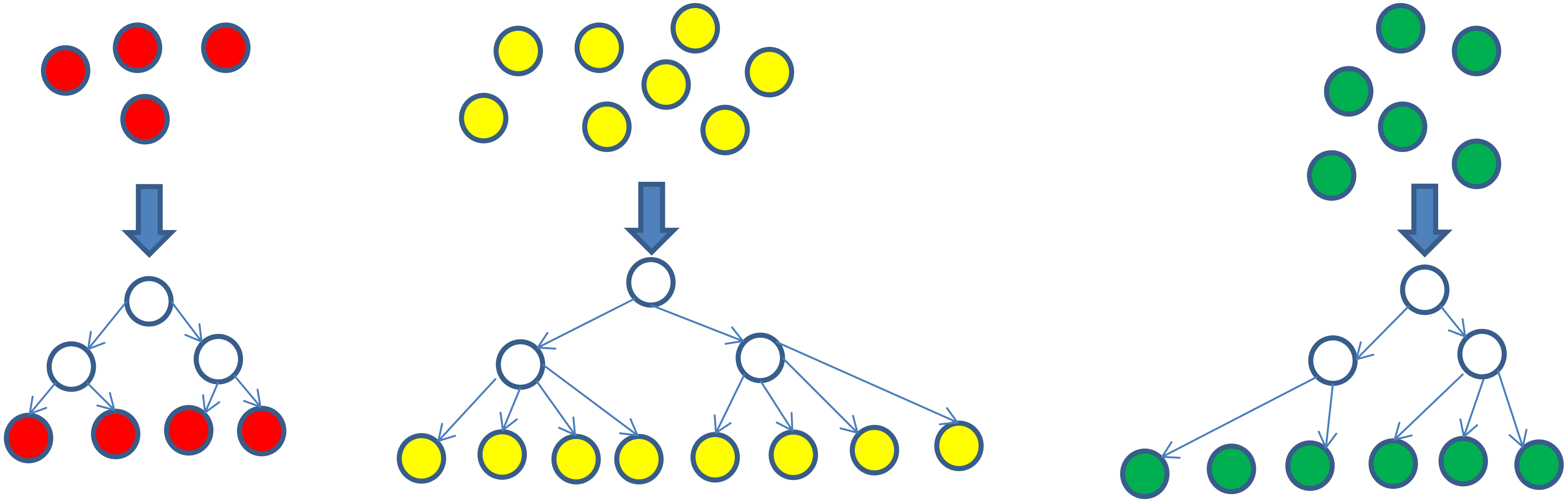
Construct an **Abstract Tree** for each individual component



Any non-leaf node **father** should be an over-approximation of his **sons** In the sense that

(… … **father** … …) sat F ➔ (… … **any son** … …) sat F

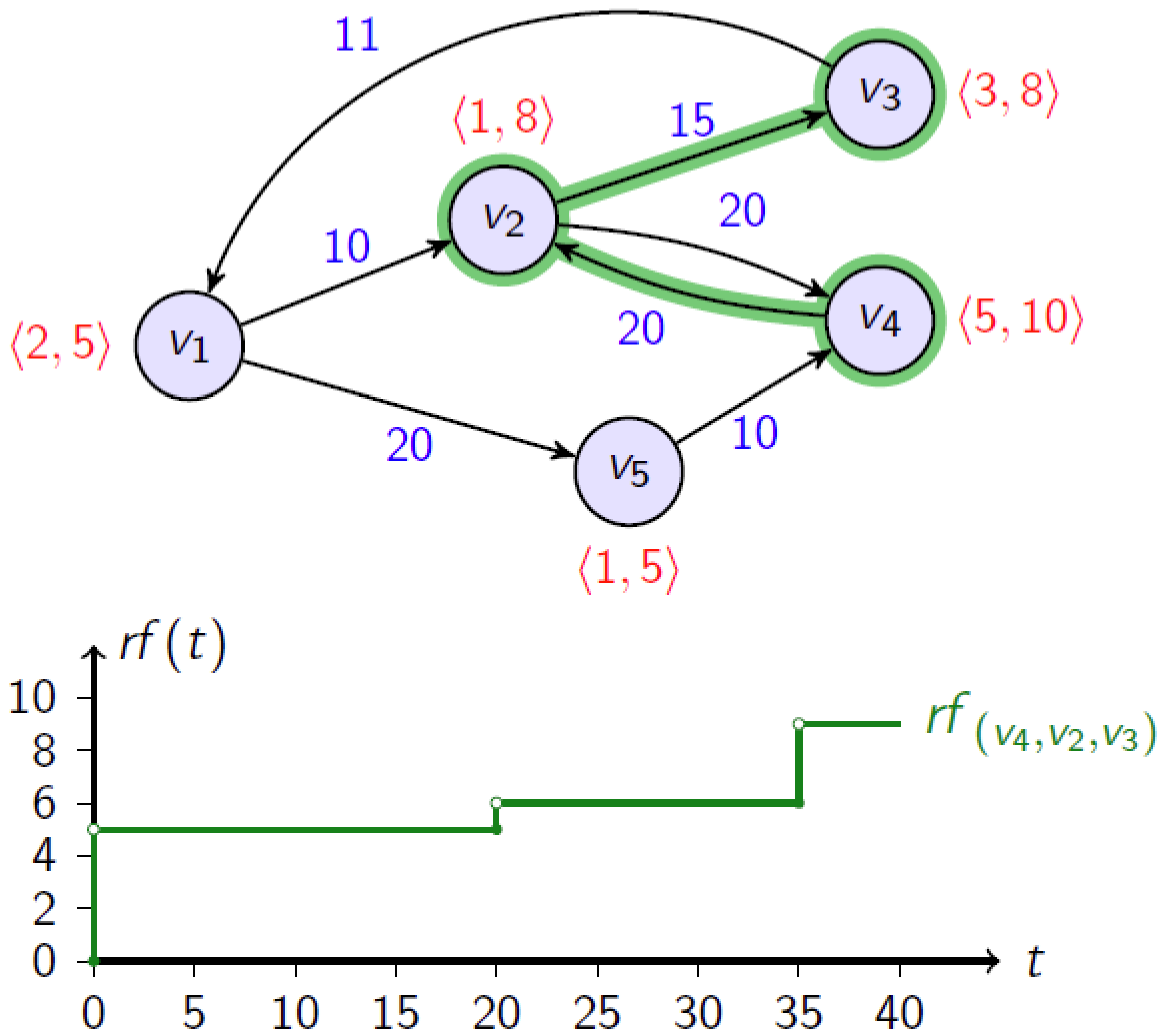
Construct an **Abstract Tree** for each individual component



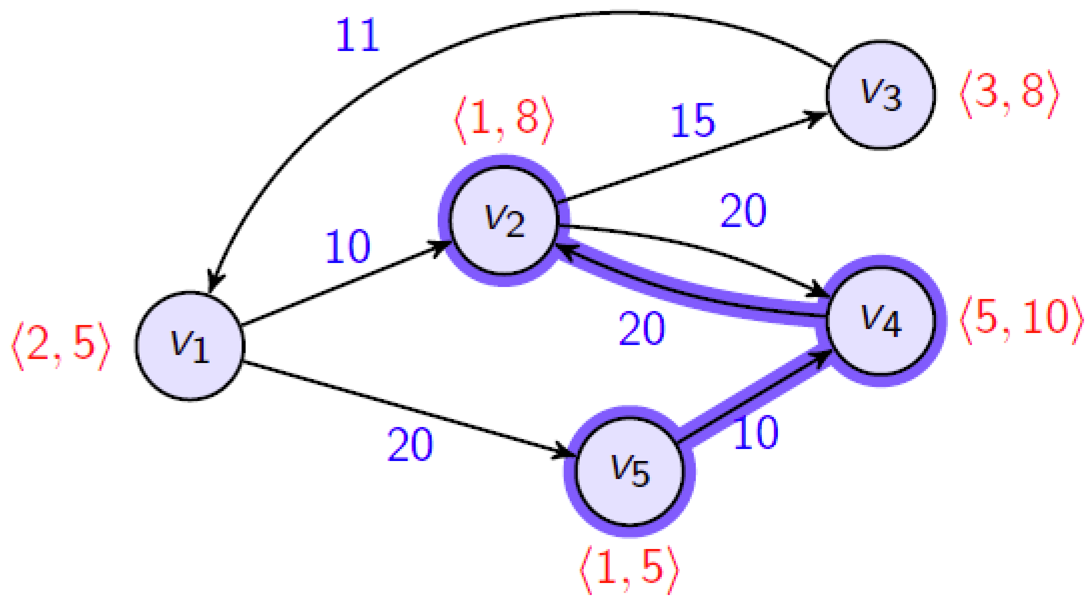
For instance, the Combination of all roots satisfies the desired property implies that all combinations of the leaves satisfy the same property.

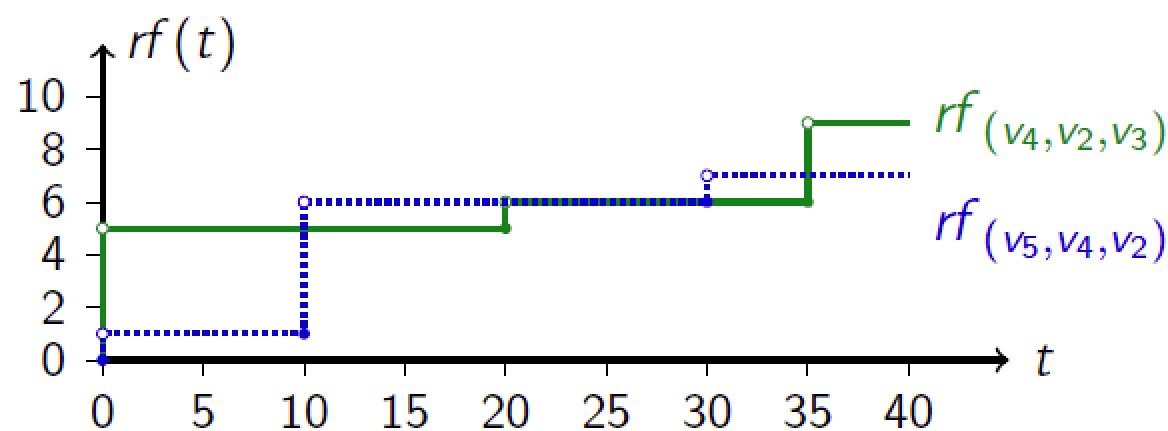
(**roots**) sat F $\rightarrow$ (**any leave, any leave, … any leave**) sat F

Abstract Request Functions

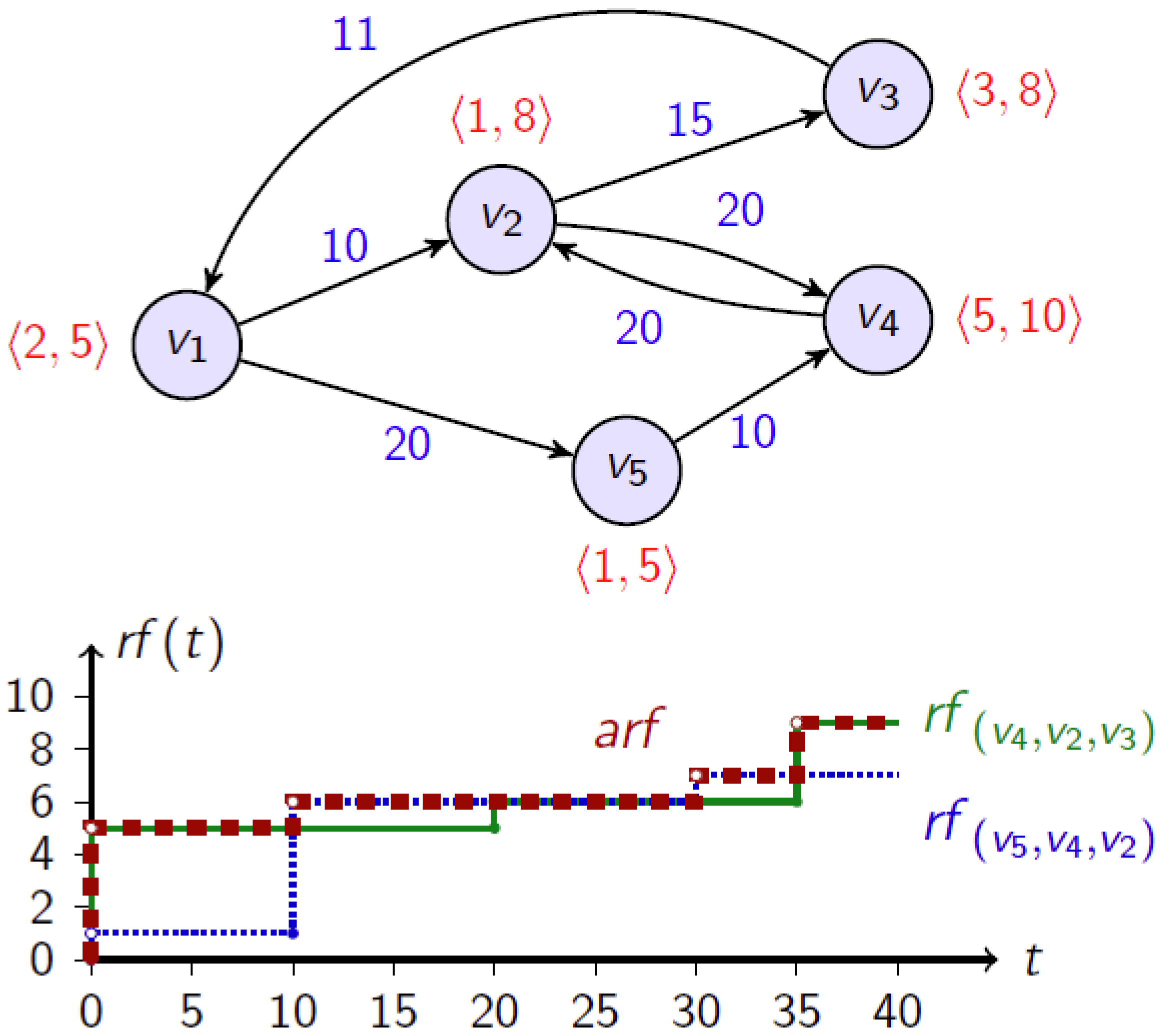


Abstract Request Functions

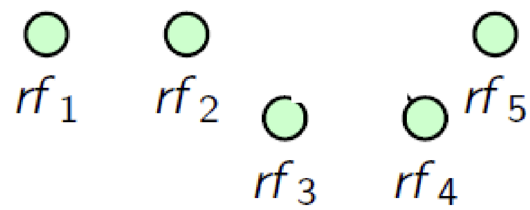




Abstract Request Functions



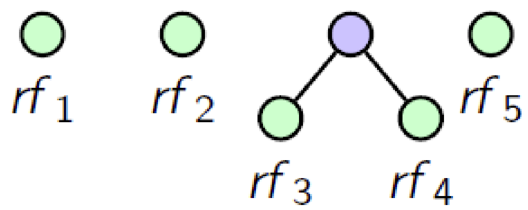
Abstraction Tree for each DRT



Define an *abstraction tree* per task:

- Leaves are concrete *rf*
- Each node: maximum function of child nodes
- Root is maximum of *all* *rf*

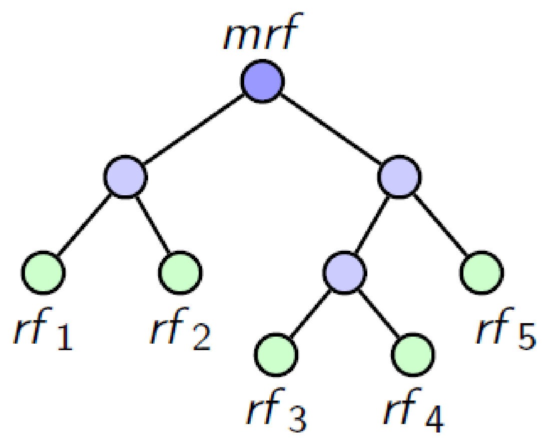
Abstraction Tree for each DRT



Define an *abstraction tree* per task:

- Leaves are concrete *rf*
- Each node: maximum function of child nodes
- Root is maximum of *all* *rf*

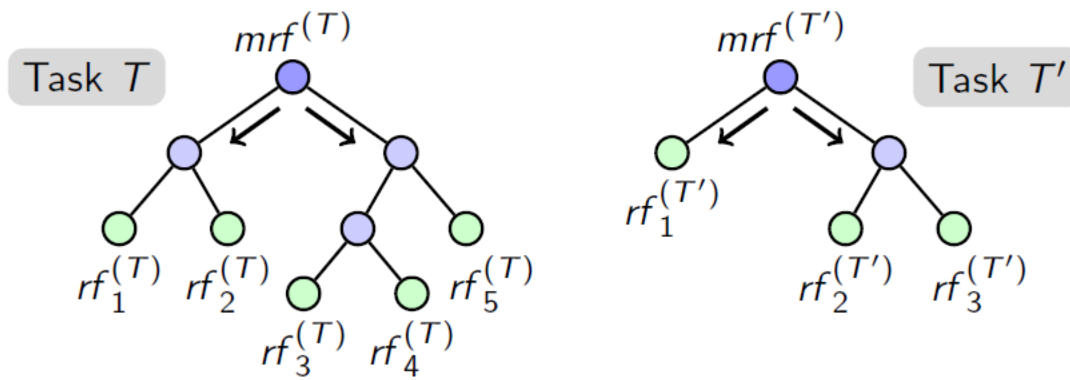
Abstraction Tree for each DRT



Define an *abstraction tree* per task:

- Leaves are concrete *rf*
- Each node: maximum function of child nodes
- Root is *mrf*, maximum of *all* *rf*

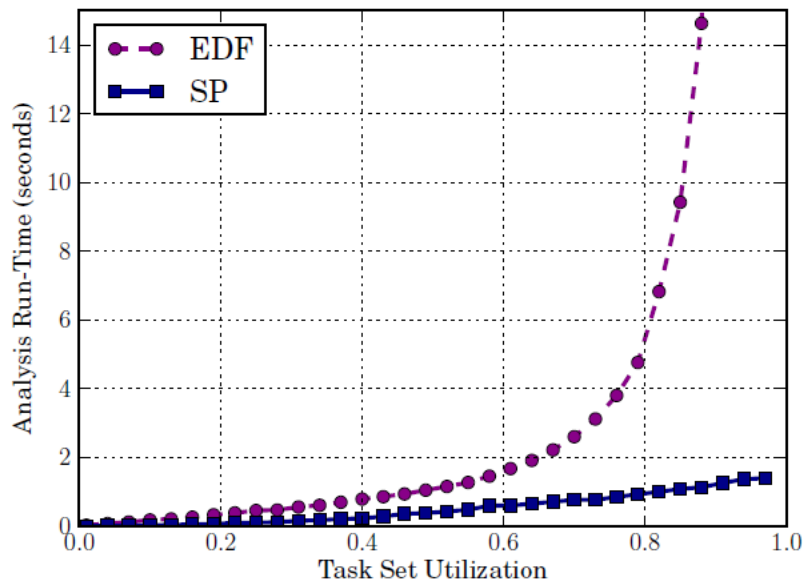
Combinatorial Abstraction Refinement



New Algorithm:

- Test *one* combination of all *mrf*.
- If fp-feasible: done
- Otherwise: Replace *one* *mrf* with all child nodes, get 2 new combinations to test
- Repeat until:
  - All combinations show fp-feasibility, or
  - A combination of leaves shows non-fp-feasibility

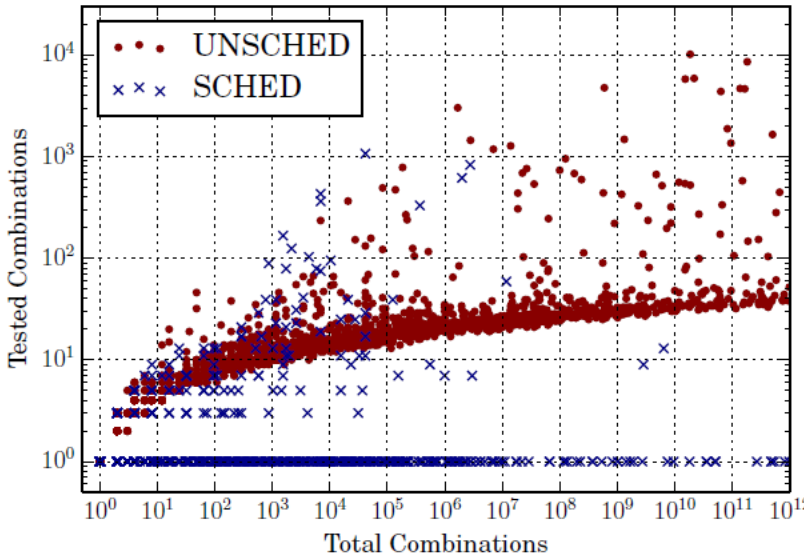
Evaluation: Runtime vs. Utilization



Comparing runtimes of

- EDF-test using dbf (pseudo-polynomial)
- SP-test based on *Combinatorial Abstraction Refinement*

Evaluation: Tested vs. Total Combinations



10^5 samples of single-job tests.

- Executed tests: in 99.9% of all cases, less than 100
- Total combinations possible: up to 10^{12}

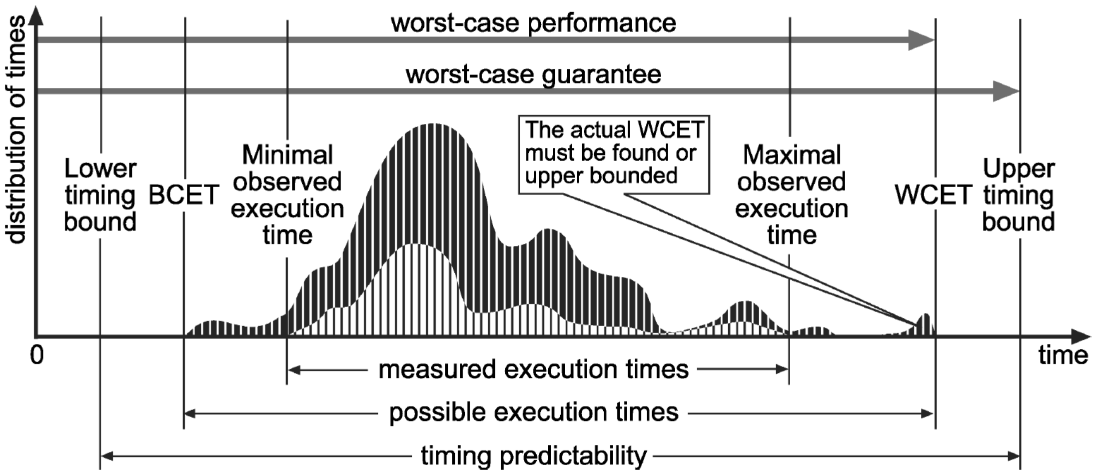
Conclusions

"Code is Art" – Daniel Licata

- Model is "Abstract Art" , the key for scalable and precise analysis
  - it should be as simple as possible but not simpler
  - it should be as expressive as possible but not more
- Digraph Model instead of Timed Automata?
  - Expressive enough to capture Ada tasking
  - Efficient analysis possible: Pseudo-polynomial
- Combinatorial Refinement works well for timing problems
  - In particular when local search space can be abstracted & ordered
  - other verification problems?
- Current work
  - Synchronization and resource sharing
  - Multiprocessor mapping and scheduling
  - TIMES++, a new tool based on Digraph, aiming at industrial applications

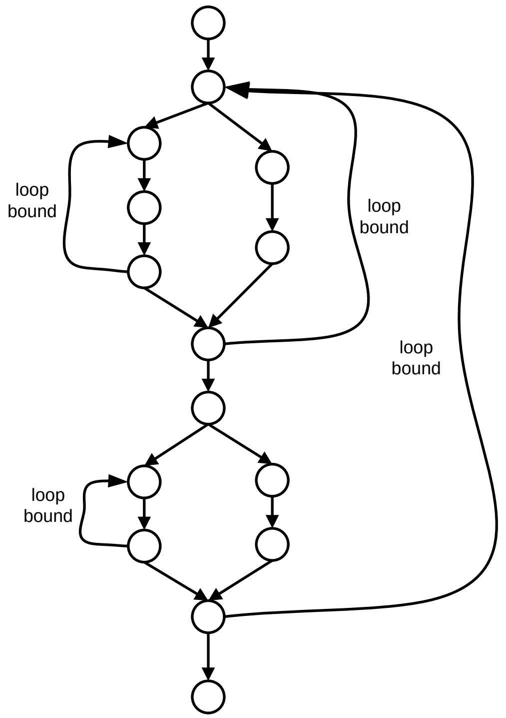
The WCET Analysis Problem

- A fundamental problem for embedded systems design
  - Worst-Case Execution Time (WCET) analysis
- Challenges ("termination" doesn't make the problem easy)
  - "too many input" ➔ too many execution paths (difficult to find the worst-case)
  - hardware features e.g. caches ("the HW state" results in different execution times)



WCET Analysis

- Path Analysis
  - which path leads to the WCET ?
  - well-known technique by ILP
  - need to know the timing delay of each instruction
- Architecture Analysis
  - Cache Analysis: Is a memory access hit or miss?
  - other factors like pipeline …



WCET Analysis

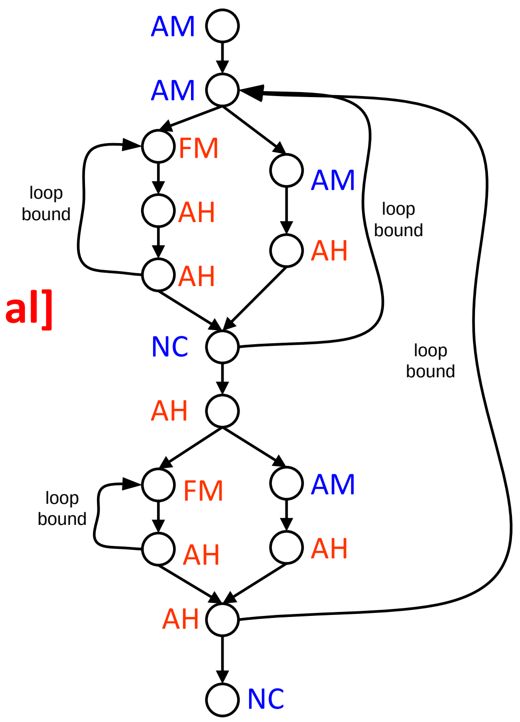
- Path Analysis
  - which path leads to the WCET ?
  - well-known technique by ILP

  - need to know the timing delay of each instruction

- Architecture Analysis **[Survey 2015 wang et al]**
  - Cache Analysis:

    Is a memory access hit or miss?
    - AH: always hit
    - FM: first miss, then always hit
    - AM: always miss
    - NC: not classified

  - other factors like pipeline …

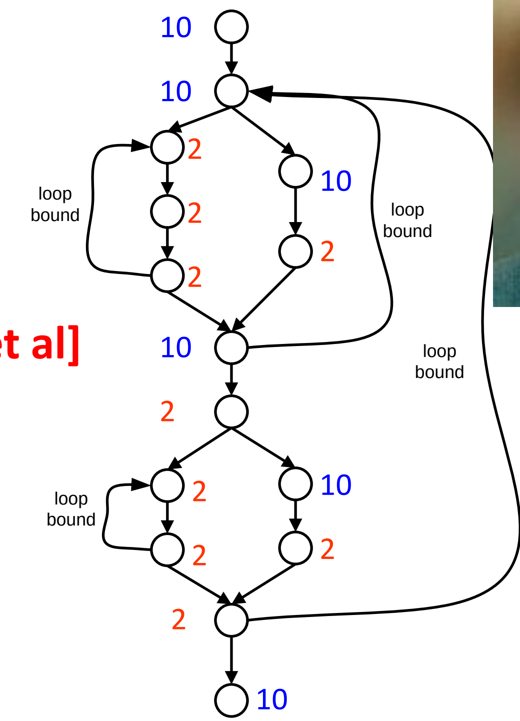


WCET Analysis

[aiT tool from AbsInt]

Wilhelm et al
Precision >> 95%

- Path Analysis
  - which path leads to the WCET ?
  - well-known technique by ILP
  - need to know the timing delay of each instruction
- Architecture Analysis **[Survey 2015 wang et al]**
  - Cache Analysis:
    Is a memory access hit or miss?
    - AH: always hit
    - FM: first miss, then always hit
    - AM: always miss
    - NC: not classified → always miss
  - other factors like pipeline …



Timing Analysis

Sequential Case (WCET Analysis)

- **Assume the WCET of each task is given (resource budget)**
- **How to estimate the Worst-Case Response Time of a task?**

Concurrent Case (Response Time Analysis)