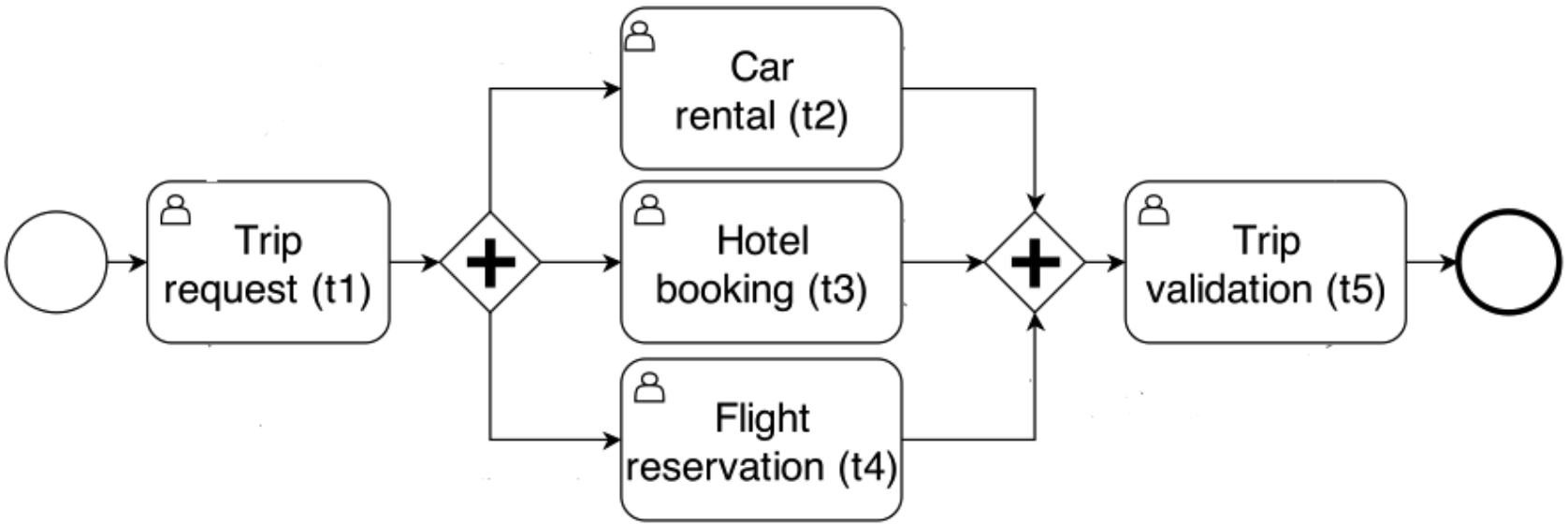
On Run-time Enforcement of Authorization Constraints in Security-Sensitive Workflows

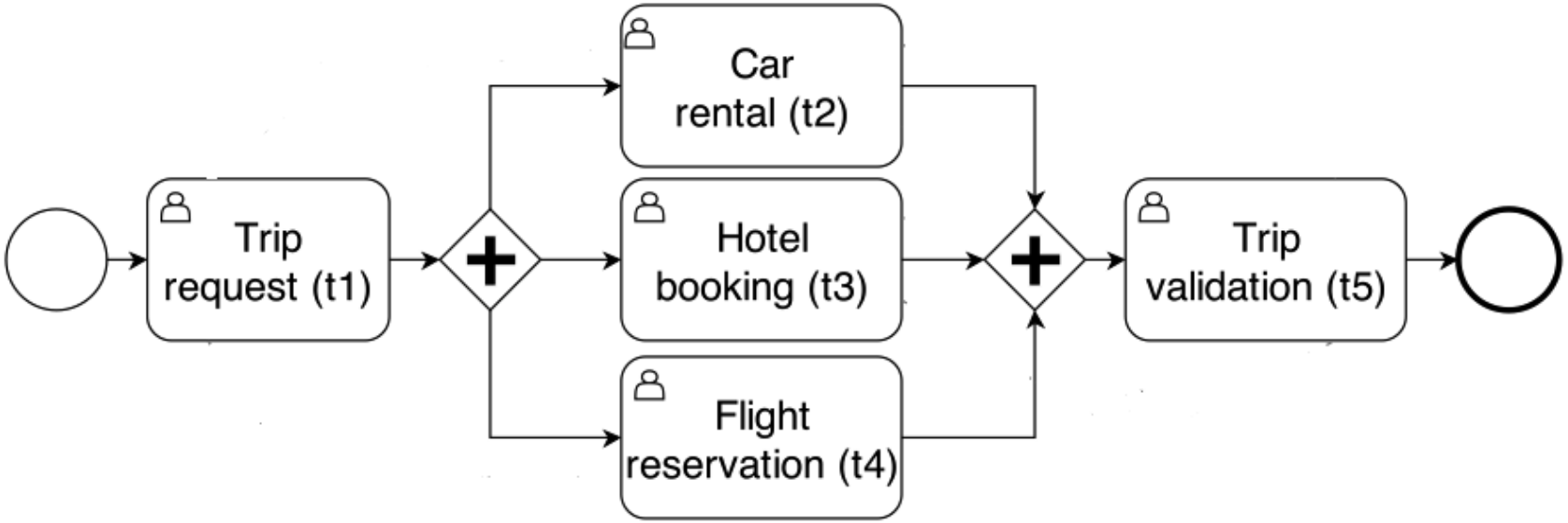
Daniel Ricardo dos Santos[1] and **Silvio Ranise**[2]

[1]TU/e and [2]FBK

Context: workflow



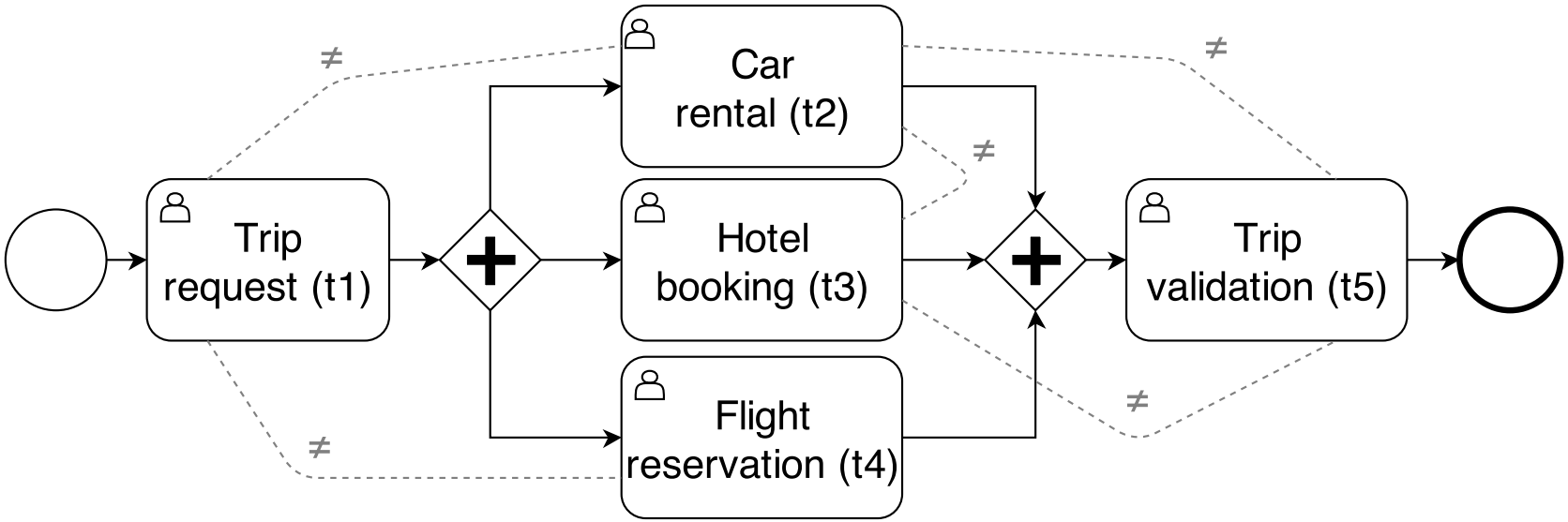
Context: workflow + policy



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Context: workflow + policy + constraints



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Problems of interest

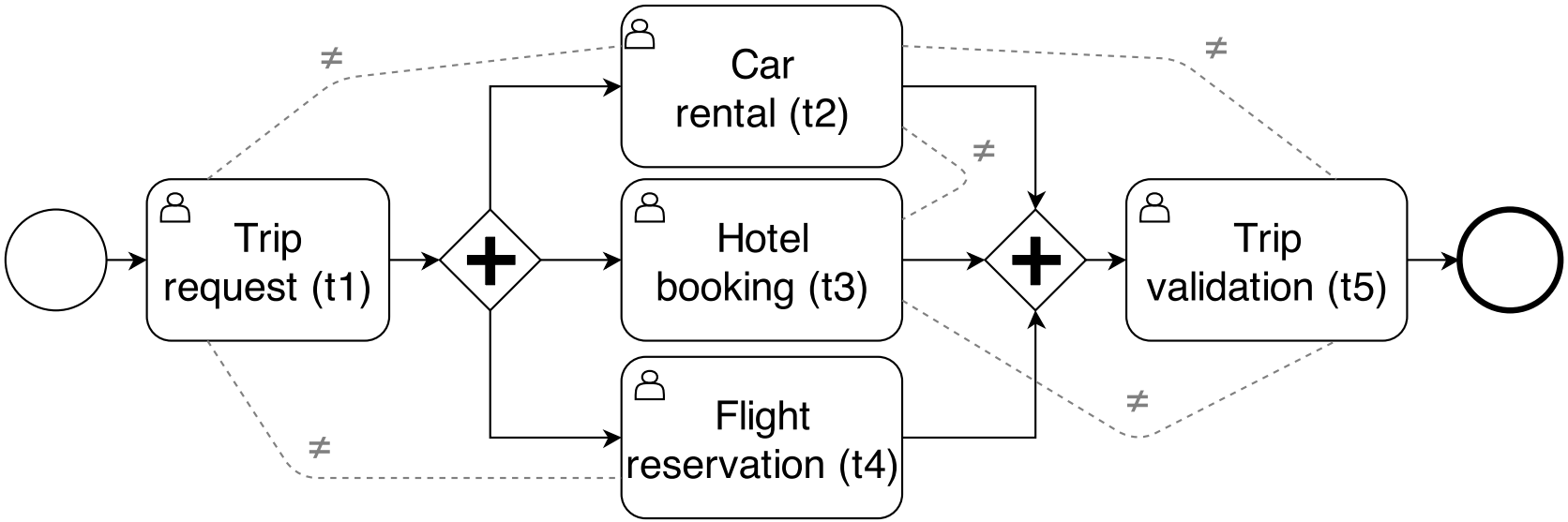
Workflow Satisfiability Problem (WSP)

Is there an assignment of users to tasks such that workflow terminates while satisfying authorization policy and constraints?

Run-time WSP

Solving sequences of WSPs induced by sequences of user requests at workflow execution time

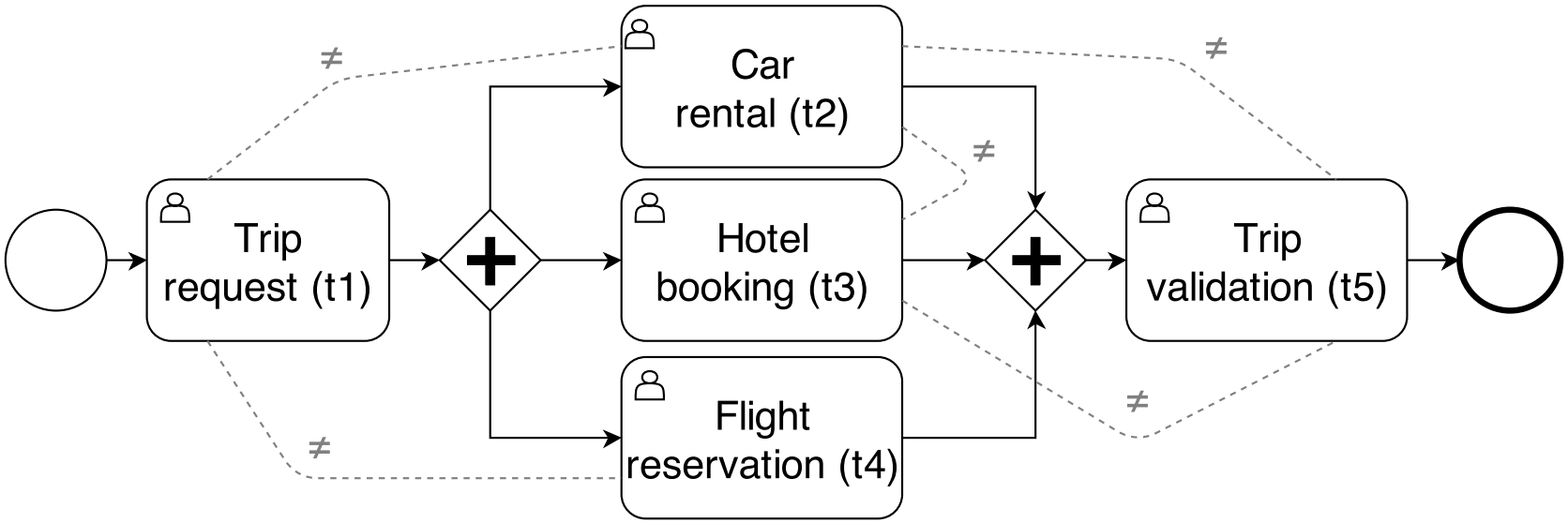
$\Longrightarrow$ focus on **Run-time** WSP



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

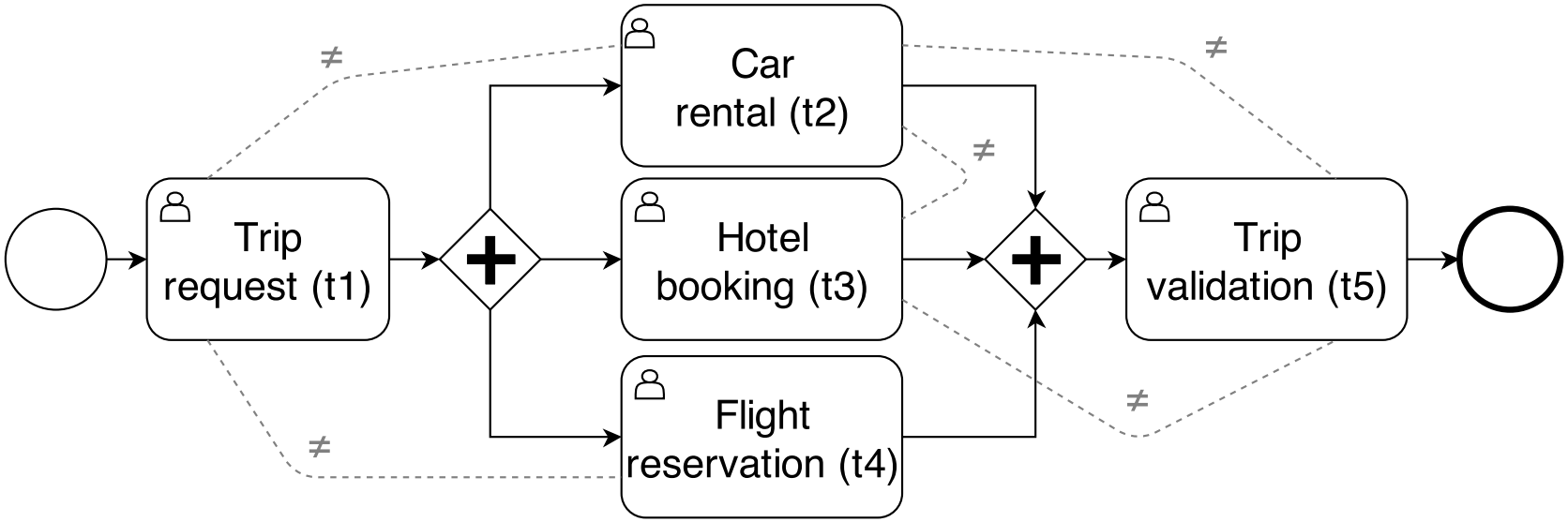
A solution to the WSP: *t1(b), t2(a), t3(c), t4(a), t5(b)*



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

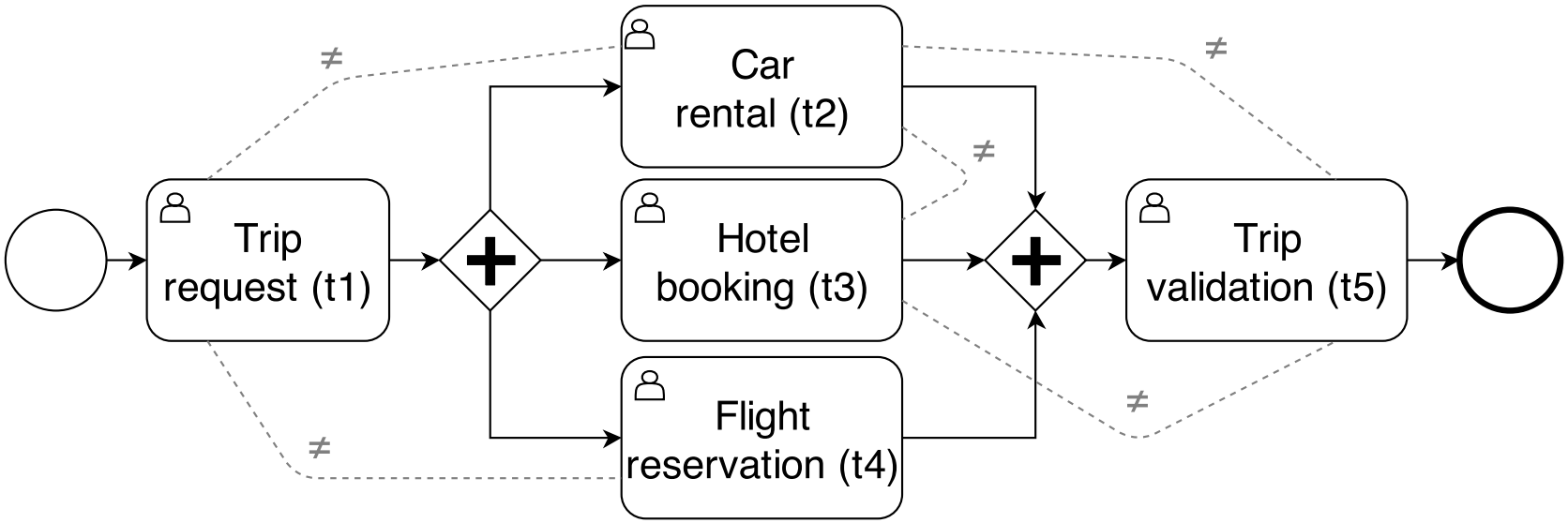
Not a solution: *t1(b), t5(b)* $\Longrightarrow$ causal relation not ok*!*



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

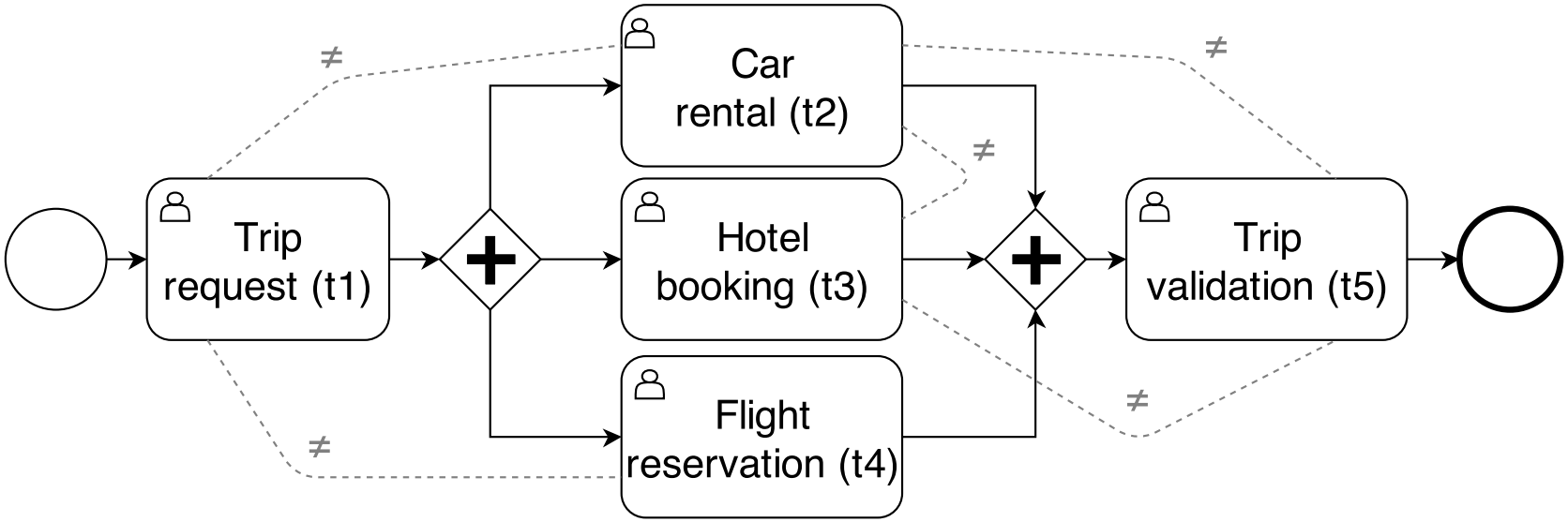
Not a solution: *t1(b), t2(a), t3(c), t4(b)* $\Longrightarrow$ policy not ok*!*
(present)



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Not a solution: *t1(b), t2(a), t3(a)* $\Longrightarrow$ *SoD*(t2,t3) not ok*!* (past)



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Not a solution: *t1(a), t2(b), t3(c), t4(a)* $\Longrightarrow$ *SoD*(t1,t4) not ok*!*
(future)

Key problems

- How can we guarantee execution of all permitted task interleavings*?*

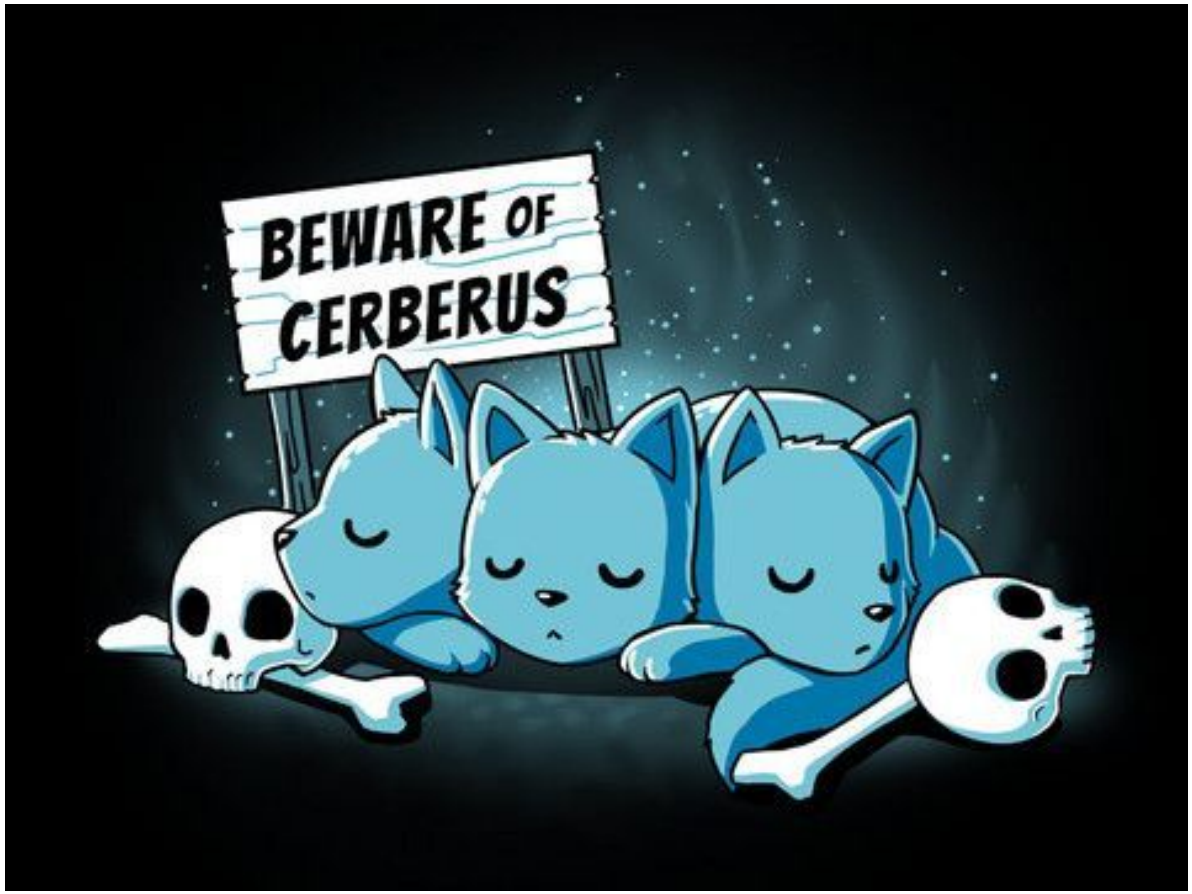
- How can we enforce an authorization policy*?*

  Present
- How can we enforce an authorization constraint*?*

  Past
- How can we enforce an authorization constraint*?*

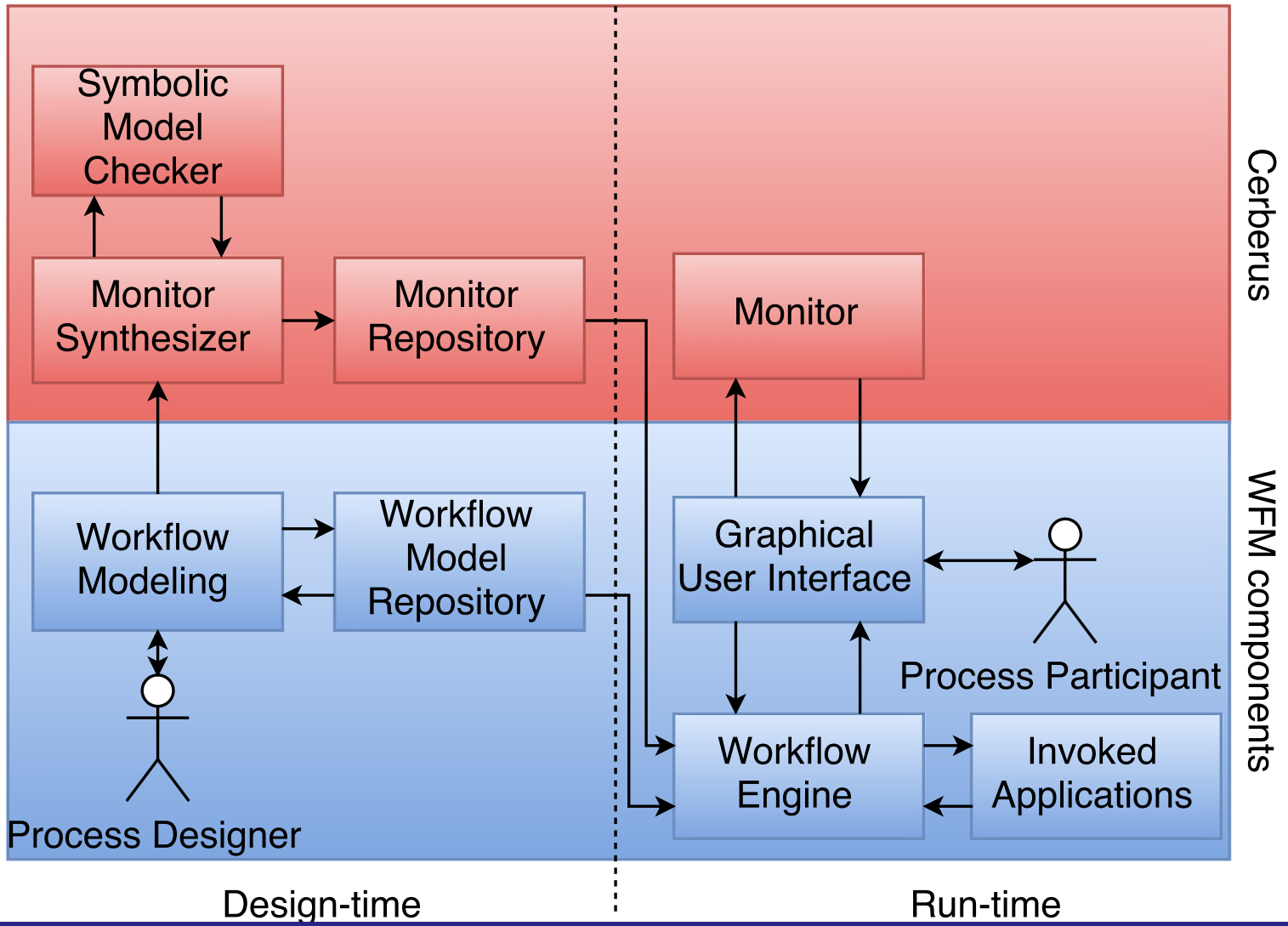
  Future

Ask Cerberus



In Greek mythology, the heads have represented Cerberus' psychic ability to *see into the past, present, and future*

Our Cerberus: architecture



Our Cerberus: key features

- Symbolic (**SMT** based) model checker allows for
  - avoiding to solve from scratch each WSP induced by the sequence of authorization queries ...
  - ... by pre-computing all possible interleavings (Petri net semantics of business processes)

- Synthezied monitors ...
  - ... can be implemented with standard and widely avaiable database technology ...
  - ... by using the equivalence between SMT fragment (Datalog) and SQL

- Scalable and **integrated in industrial environment**
  (SAP Hana as SAP was industrial partner in SECENTIS)

Our Cerberus: strengths over other solutions

- Almost all solutions focus on WSP
  (mainly work by *Crampton et al*)
- *Basin-Burri-Karjoth* offer only an approximate solution to run-time WSP
- Not integrated in industrial contexts

Our Cerberus: open issues

So far considered **only** Separation/Binding of Duty constraints*!*

- Which one of the constraint classes considered in the literature Cerberus is able to support*?*

  $\Longrightarrow$ reduced scope of applicability*!*

- *Preliminary*: no systematic study of classes of authorization constraints in the literature is available

This work: contributions

1. Systematic classification of known classes of constraints
   $\Longrightarrow$ with relationships in terms of expressivity

2. Encodings of several classes of constraints in Cerberus
   $\Longrightarrow$ thereby significantly enlarging its scope of applicability

This work: contributions

1. Systematic classification of known classes of constraints
   $\Longrightarrow$ with relationships in terms of expressivity

2. Encodings of several classes of constraints in Cerberus
   $\Longrightarrow$ thereby significantly enlarging its scope of applicability

Classification of authorization constraints (I)

- **Counting**: (t_l, t_r, T) with $1 \leq t_l \leq t_r \leq k$

  Meaning: a user shall perform either no tasks in T or between t_l and t_r tasks

- **Entailment**: (T_1, T_2, ρ) with $\rho \subseteq U \times U$

  Meaning: there shall exist $t_1 \in T_1$ and $t_2 \in T_2$ s.t. $(u_1, u_2) \in \rho$ when u_i has executed t_i for $i = 1, 2$

  Sub-class: **Equivalence** when ρ is an equivalence

Classification of authorization constraints (II)

- **User-independent**: examples are Separation of Duties

  Intuition: satisfaction does not depend on user identity

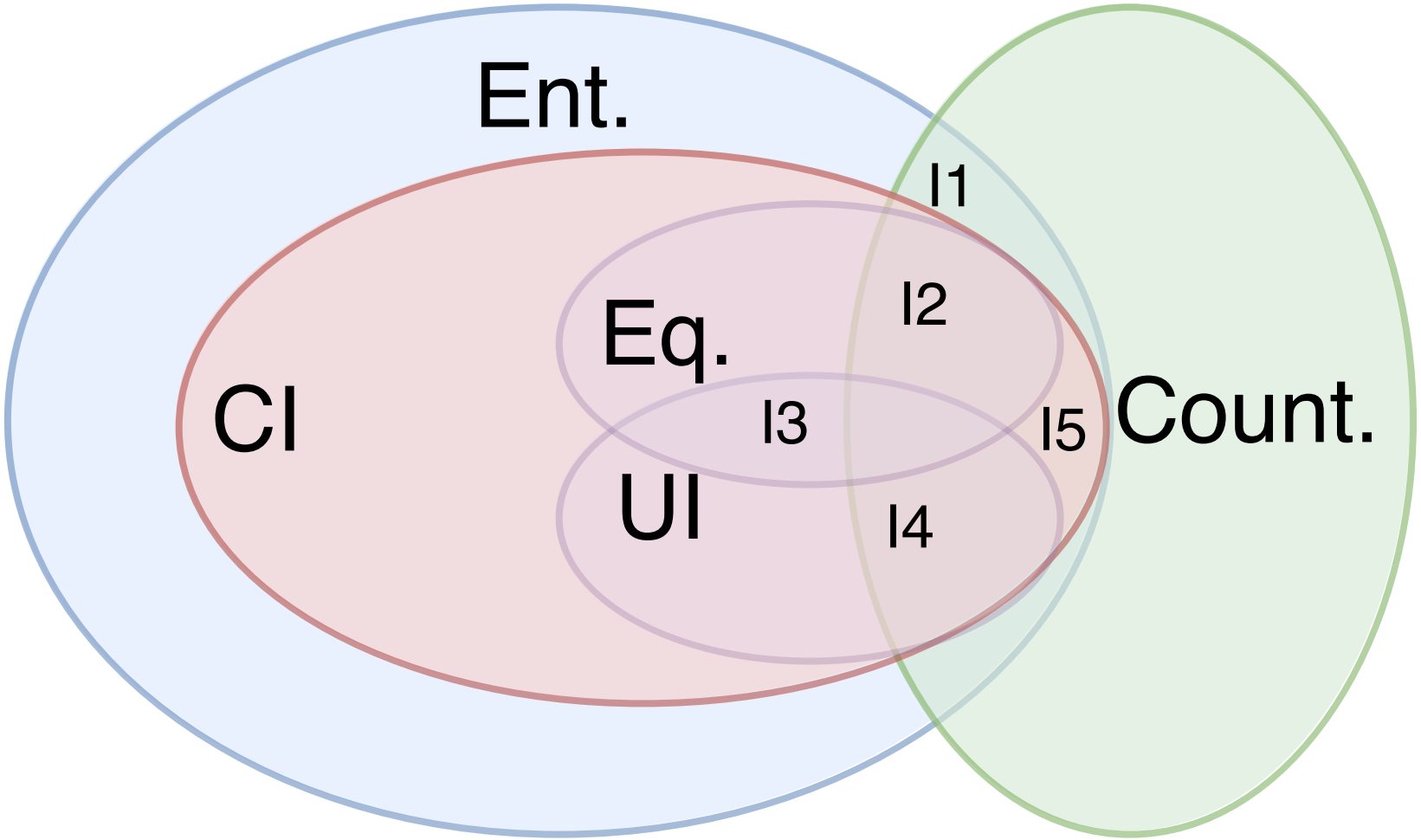
  Formalization by permutation of users

- **Class-independent**: examples are certain entailment constraints such as $(\{t_1\}, \{t_2\}, \sim)$ where equivalence classes of $\sim$ are the departments of an organization

  Intuition: satisfaction depends only on the equivalence classes that users belong to

  Formalization by permutation of equivalence classes of users

Classification of authorization constraints (III)



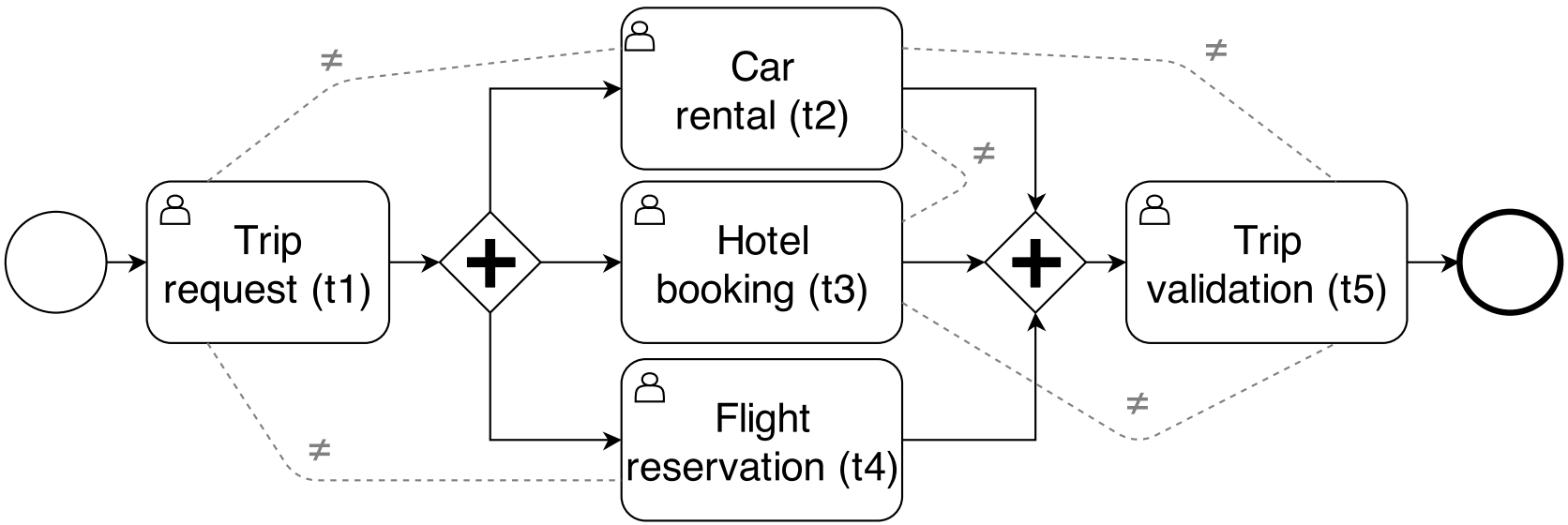
This work: contributions

1. Systematic classification of known classes of constraints
   $\Longrightarrow$ with relationships in terms of expressivity

2. Encodings of several classes of constraints in Cerberus
   $\Longrightarrow$ thereby significantly enlarging its scope of applicability

Anatomy of a transition in Cerberus

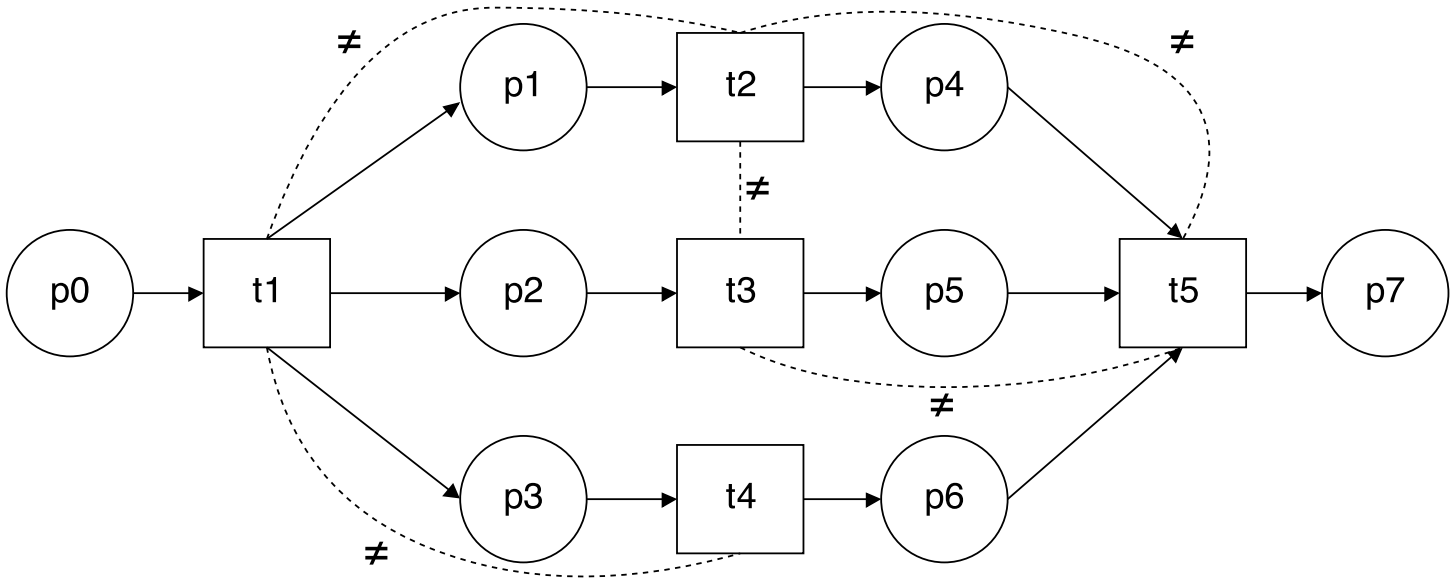
$$\text{t2(u): } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t1}(u)$$
$$\longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$



Anatomy of a transition in Cerberus: explained

$$\text{t2(u): } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t1}(u)$$
$$\longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

Based on Petri net semantics of Business Processes:



Anatomy of a transition: crucial property

$$\text{t2(u): } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t1}(u)$$
$$\longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

With this format of transitions, ***model checking algorithm*** *(backward reachability) used by Cerberus is* ***guaranteed*** *to* ***terminate*** *regardless of the number of users!*

Anatomy of a transition: tricky point

$$\text{t2(u): } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \boxed{\neg h_{t3}(u) \wedge \neg h_{t1}(u)} \longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

Boxed formula is the result of a transformation as original SoD(t1,t2) and SoD(t2,t3) can be specified as $\forall w.(\neg h_{t1}(w) \wedge \neg h_{t2}(w)) \wedge (\neg h_{t2}(w) \wedge \neg h_{t3}(w))$ *and the* ***equivalent*** *transition*

$$t2(u): p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \boxed{\forall w.(\neg h_{t1}(w) \wedge \neg h_{t2}(w)) \wedge (\neg h_{t2}(w) \wedge \neg h_{t3}(w))} \longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

does ***not*** *guarantee* ***termination*** *of model checking algorithm used in Cerberus!*

Anatomy of a transition: tricky point, symbolic solution

It is possible to show that it is ***sufficient to instantiate the universal variable*** *w in*

$$t2(u)\text{: } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \boxed{\forall w.(\neg h_{t1}(w) \wedge \neg h_{t2}(w)) \wedge (\neg h_{t2}(w) \wedge \neg h_{t3}(w))} \longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

to the (implicitly existentially quantified) variable *u and derive*

$$t2(u)\text{: } p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \boxed{\neg h_{t3}(u) \wedge \neg h_{t1}(u)} \longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$$

to make the model checking algorithm used by Cerberus ***consider the same set of execution***

Anatomy of a transition: symbolic solution, scope (I)

- Many other constraints can be encoded as *universally quantified invariants* ...
- ... so that the **same symbolic transformation** can be applied to **guarantee termination** of (the model checking algorithm used by) Cerberus*!*
- Intuition: define suitable symbolic representation of constraint classes
  - **Entailment** ...
  - **Counting** ...

Anatomy of a transition: symbolic solution, scope (II)

Suitable symbolic representation of **Entailment**

- (T_1, T_2, ρ) can be specifed as

$$\forall z_1, z_2. \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} h_{t1}(z_1) \wedge h_{t2}(z_2) \Rightarrow \rho(z_1, z_2)$$

- Additionally: ρ must have a "*simple algebric structure*"

  Formally: axiomatized by *universally quantified formulae with only predicate symbols*

Anatomy of a transition: symbolic solution, scope (III)

Suitable symbolic representation of **Counting**

- (T, t_l, t_r) can be specifed as

$$\forall U_T. \left(\bigwedge_{t \in T} h_t(u_t) \Rightarrow \text{AtMost}(U_T, t_r) \wedge \text{AtLeast}(U_T, t_l) \right)$$

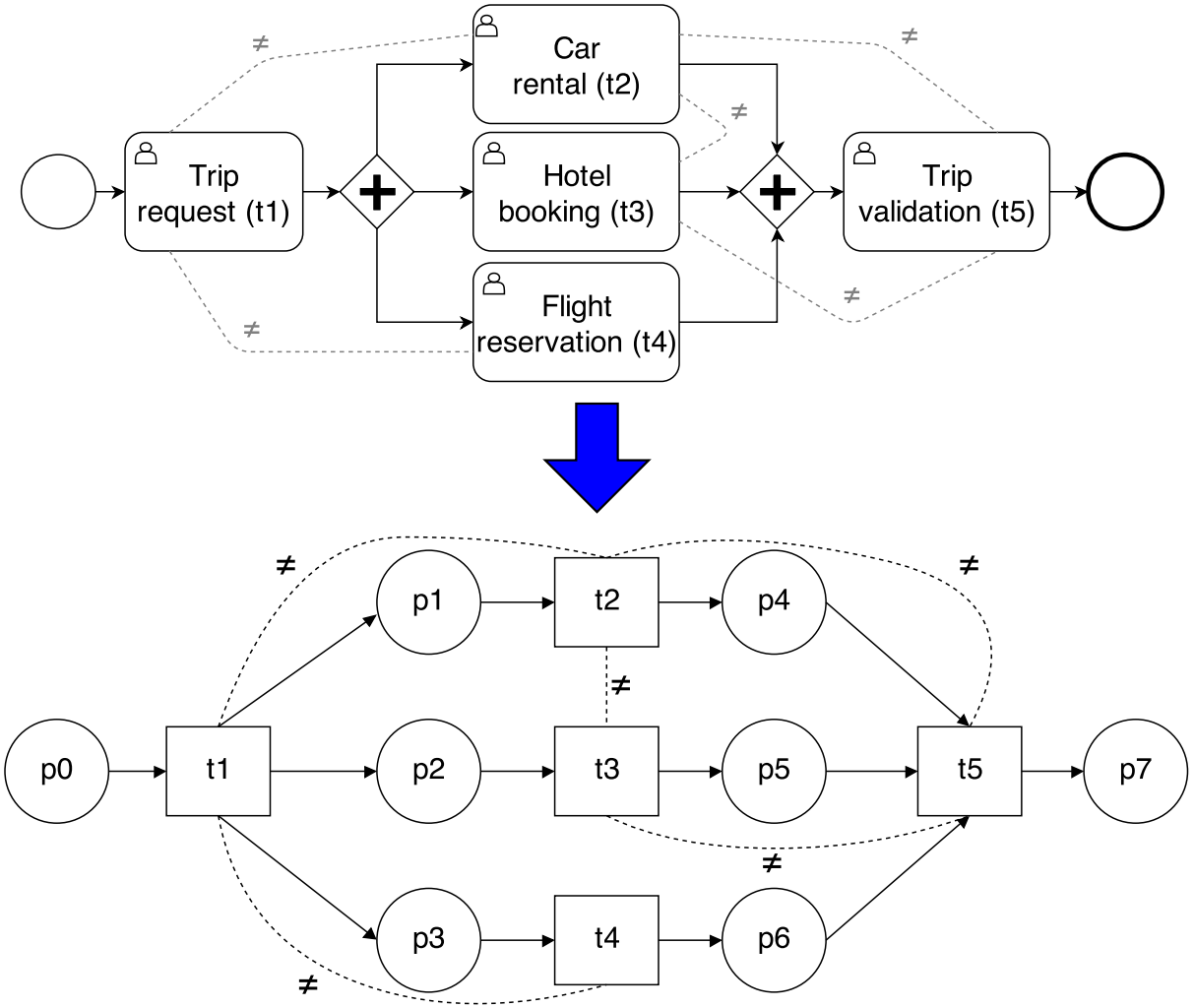
  where
  - $U_T = \{u_t | t \in T\}$
  - $\text{AtMost}(U_T, t_r) := \bigvee \forall \overline{U_T}. \bigvee_{x \neq y \in \overline{U_T}} x \neq y$ for $\overline{U_T} \subseteq U_{T'}$ of cardinality t_r
  - $\text{AtLeast}(U_T, t_l) := \bigwedge \exists \overline{U_T}. \bigwedge_{x \neq y \in \overline{U_T}} x \neq y$ for $\overline{U_T} \subseteq U_T$ of cardinality t_l

Conclusions and future work

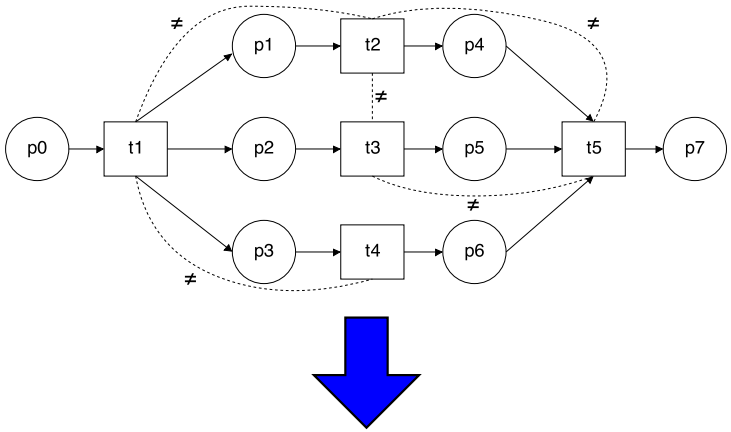
- **Contributions**
  1. Systematic classification of known classes of constraints
     $\Longrightarrow$ with relationships in terms of expressivity
  2. Encodings of several classes of constraints in Cerberus
     $\Longrightarrow$ thereby significantly enlarging its scope of applicability
- **Future work**
  - Comparison with algebra of authorization constraints proposed by Basin-Burri-Karjoth
  - Experimental evaluation with considered classes of authorization constraints

Back-up slides

BPMN to Petri Net



Petri Net to Transition System



t(u): $enabled_{CF} \wedge enabled_{Auth} \longrightarrow act_{CF} || act_{Auth}$

t2(u): $p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t1}(u)$
$\longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$

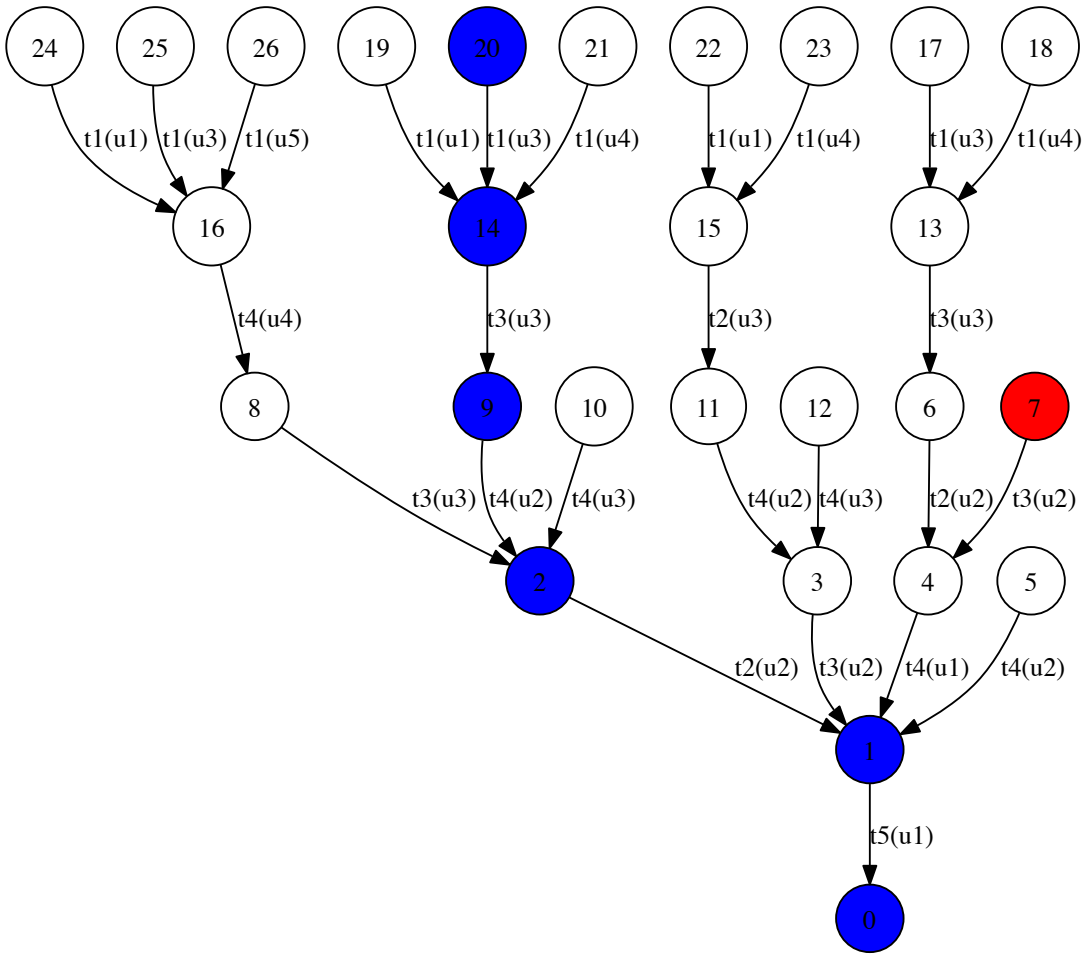
I:$p0 \wedge ... \wedge p7 \neg pi \wedge \neg d_{ti}, ..., d_{t5} \wedge \bigwedge_{i=1,...,5} \forall u. \neg h_{ti}(u)$
F:$p7 \wedge \neg p0 \wedge ... \neg p6 \wedge d_{t1} ... \wedge d_{t5}$

Computing the Reachability Graph

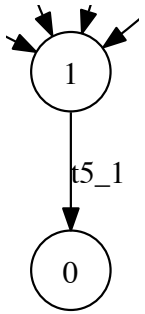
Require: $S = (V_{CF} \cup V_{Auth} \cup V_{User}, Ev_S)$ and F
Ensure: $RG = (N, \lambda, E)$

1: #start with a node for the final state
2: $i \leftarrow$ new(); $N \leftarrow \{i\}$; $E \leftarrow \emptyset$; $\lambda[i] \leftarrow F$; $TBV \leftarrow \{i\}$;
3: **while** $TBV \neq \emptyset$ **do** #until a fix-point is reached
4: **if** `subsumed`(i,N,N') **then**
5: `connect`(N',i); $TBV \leftarrow TBV - \{i\}$;
6: **end if**
7: **for all** $ev \in Ev_S$ **do** #then for every transition
8: $P \leftarrow \text{wlp}(ev, \lambda[i])$;
9: **if** P is satisfiable **then** #add a new node if applicable
10: $j \leftarrow$ new(); $N \leftarrow N \cup \{j\}$; $E \leftarrow E \cup \{(i, \overline{ev}, j)\}$;
11: $\lambda[j] \leftarrow P$; $TBV \leftarrow TBV \cup \{j\}$;
12: **end if**
13: **end for**
14: $i \leftarrow$ `pickOne`(TBV); $TBV \leftarrow TBV - \{i\}$;
15: **end while**
16: **return** (N, λ, E);

Reachability Graph



Reachability Graph to Datalog



$$\neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge (a_{t5}(u1) \wedge \neg h_{t2}(u1) \wedge \neg h_{t3}(u1))$$

$$can\_do(u1, t5) \leftarrow \neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge a_{t5}(u1) \wedge \neg h_{t2}(u1) \wedge \neg h_{t3}(u1)$$

Policy

RBAC Policy

$U = \{a, b, c\}$ $R = \{r_1, r_2, r_3\}$
$UA = \{(a, r1), (a, r2), (a, r3), (b, r2), (b, r3), (c, r2)\}$
$TA = \{(r_3, t1), (r_2, t2), (r_2, t3), (r_1, t4), (r_2, t5)\}$

Policy in Datalog

$ua(a, r1)$ $ua(a, r2)$ $ua(a, r3)$ $ua(b, r2)$ $ua(b, r3)$ $ua(c, r2)$
$pa(r_3, t1)$ $pa(r_2, t2)$ $pa(r_2, t3)$ $pa(r_1, t4)$ $pa(r_2, t5)$
$a_t(u) \leftarrow ua(u, r) \wedge pa(r, t)$ for each $t \in \{t1, ..., t5\}$

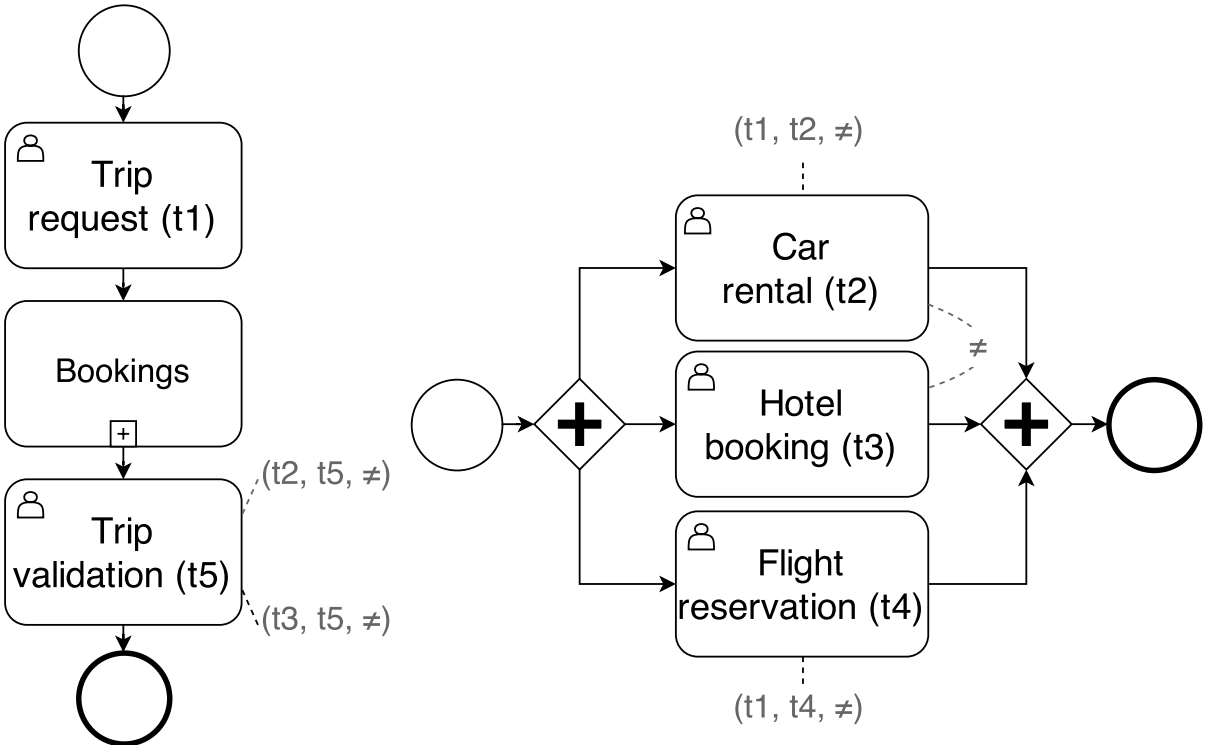
Example trace

	CF	Auth					*can_do*	
#	Token in	h_{t1}	h_{t2}	h_{t3}	h_{t4}	h_{t5}	(u, t)	Resp.
0	$p0$	-	-	-	-	-	$(a, t1)$	deny
1	$p0$	-	-	-	-	-	$(b, t1)$	grant
2	$p1, p2, p3$	b	-	-	-	-	$(b, t2)$	deny
3	$p1, p2, p3$	b	-	-	-	-	$(a, t2)$	grant
4	$p4, p2, p3$	b	a	-	-	-	$(c, t3)$	grant
5	$p4, p5, p3$	b	a	c	-	-	$(a, t4)$	grant
6	$p4, p5, p6$	b	a	c	a	-	$(b, t5)$	grant
7	$p7$	b	a	c	a	b	-	-

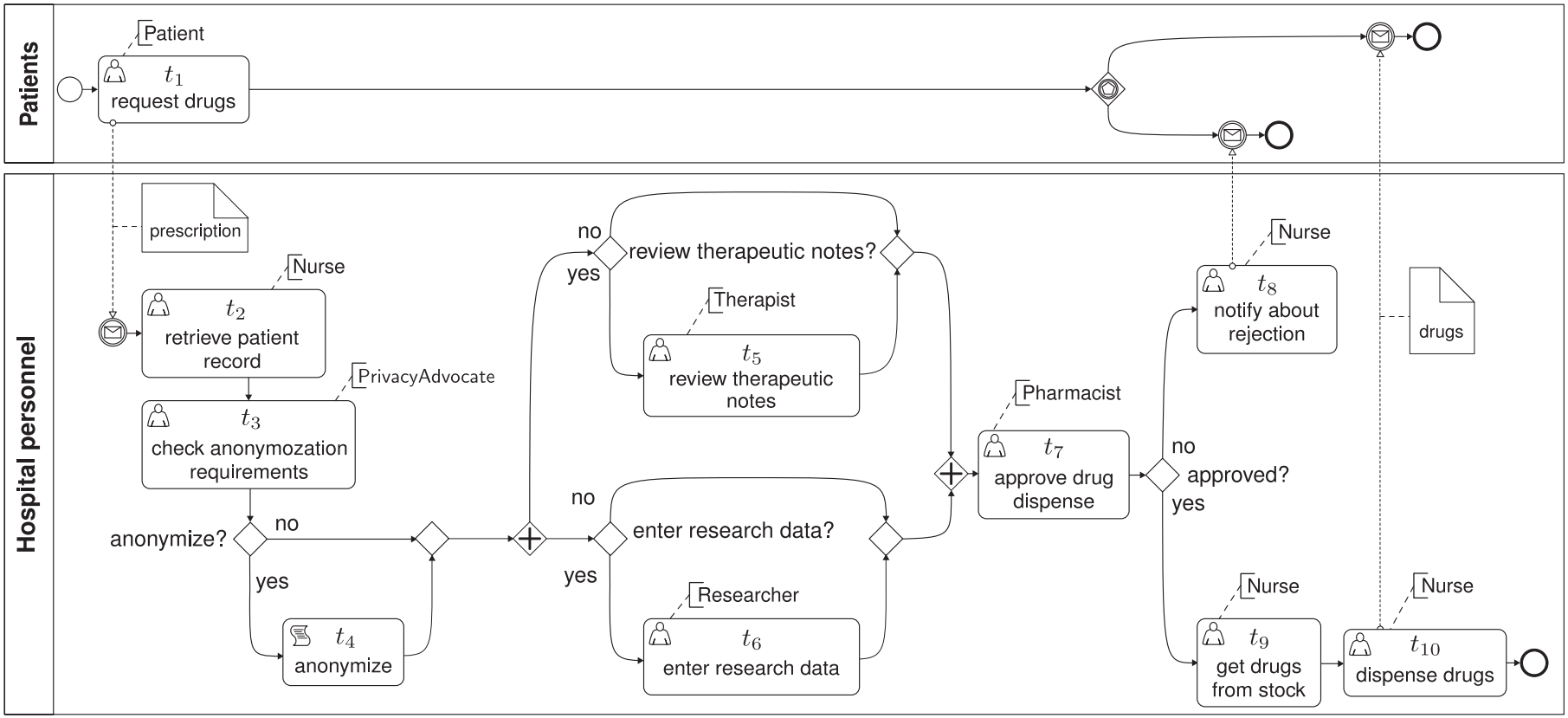
Description

- Real-world for validity
  - Moderate number of tasks
  - Complex control-flow
- Synthetic for scalability
  - Huge number of tasks
  - Simple control-flow
- State space explosion $\rightarrow$ need for heuristics

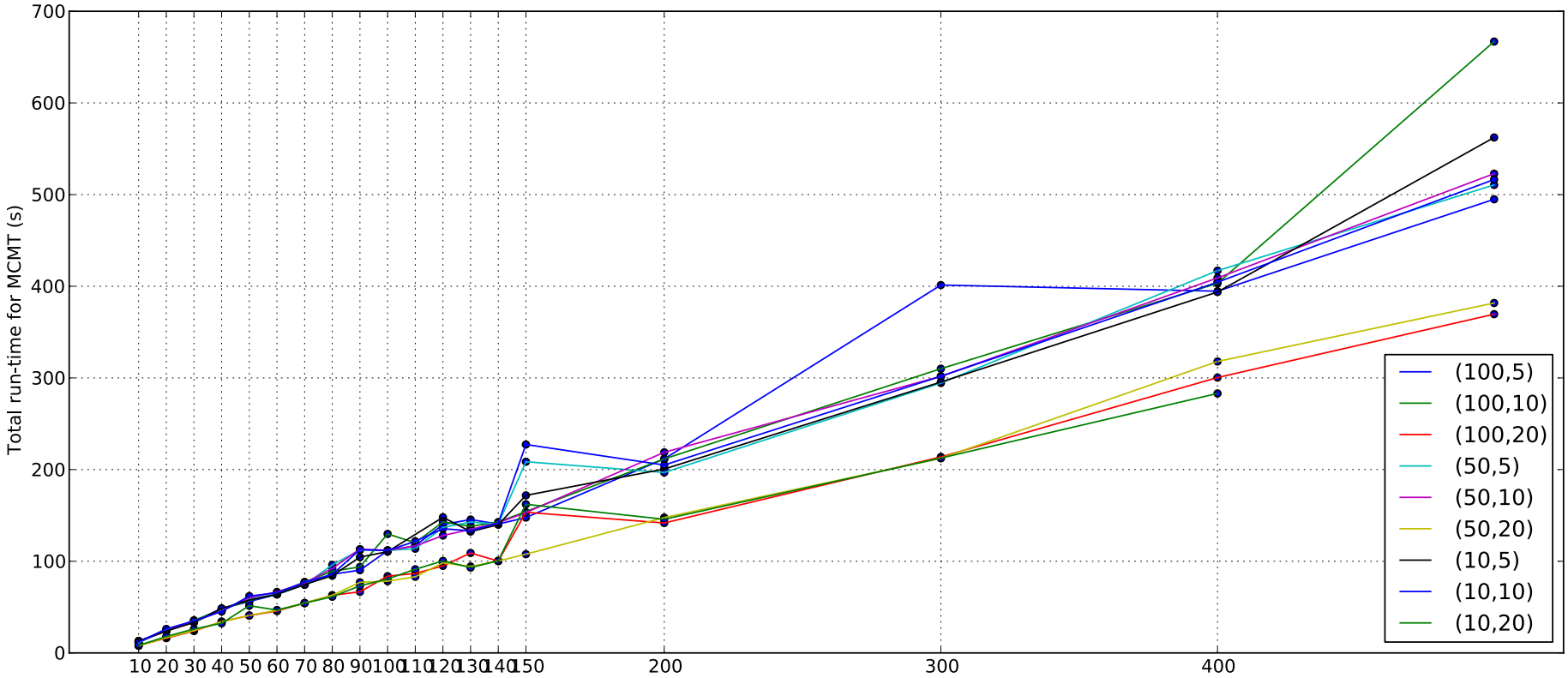
Hierarchical descriptions



Drug dispensation process

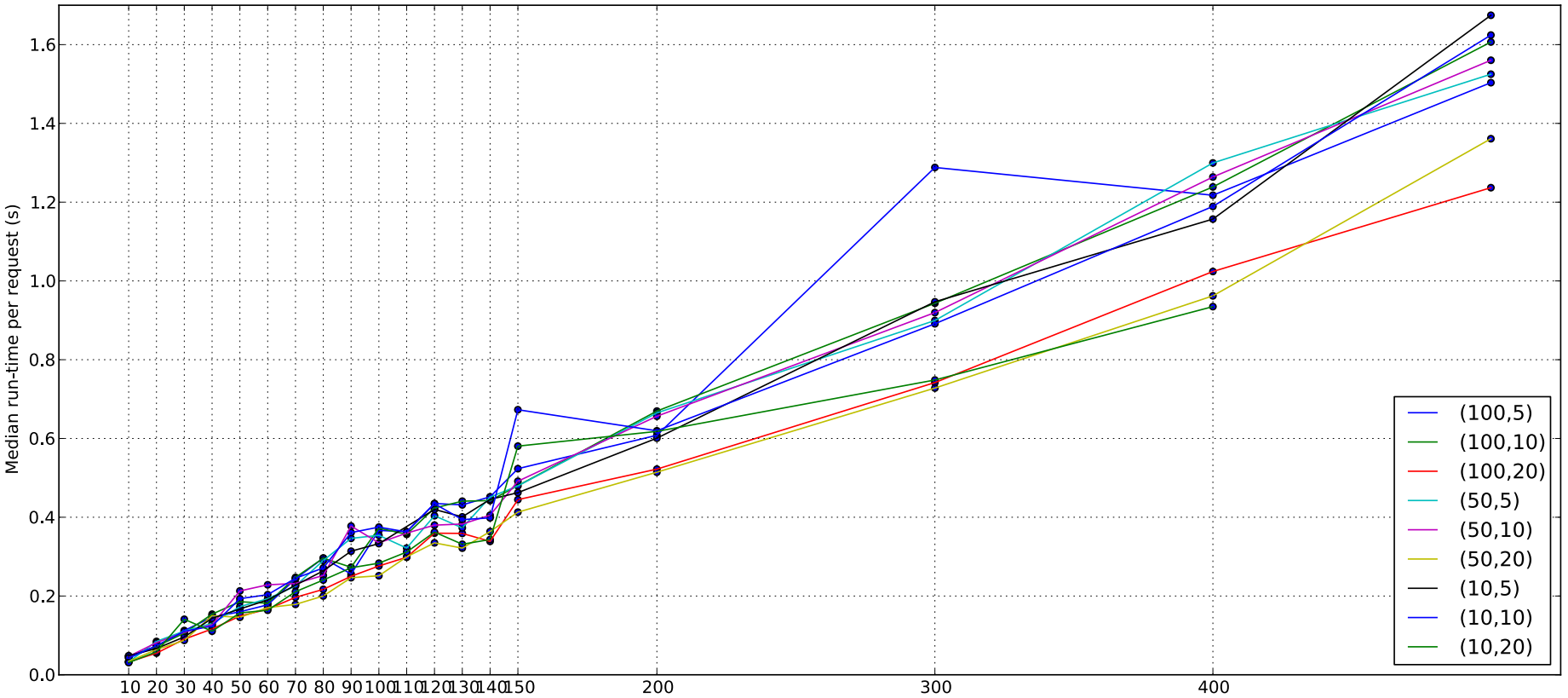


Off-line



(pa,pc) = (authorization density, constraint density)

On-line



Number of tasks

(pa,pc) = (authorization density, constraint density)