Real-Time BIP

Jacques Combaz

January 30, 2014

Real-Time Extension of BIP

BIP model (reminder):

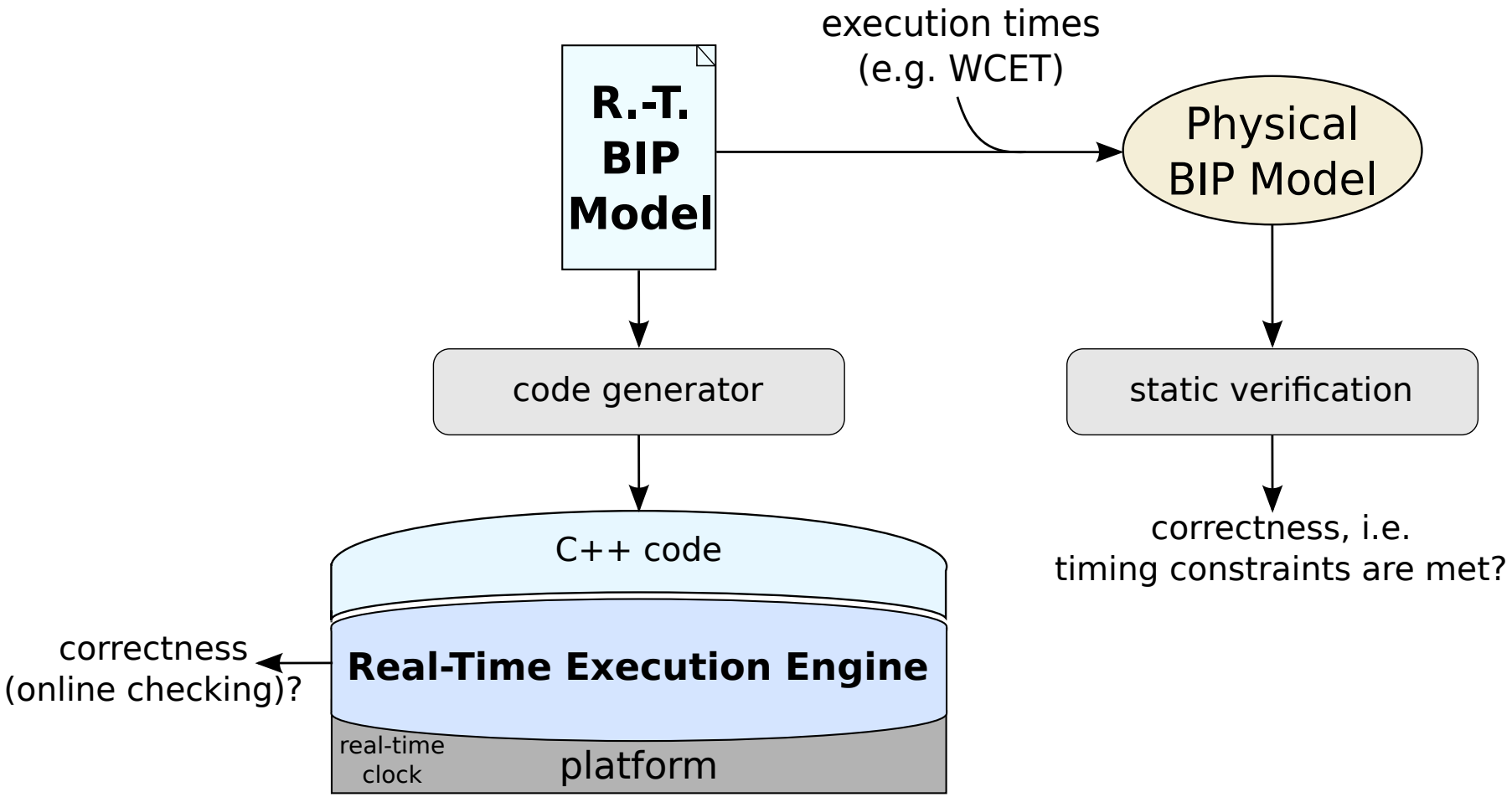
- **B**ehavior—atomic components: automata + ports + data
- **I**nteractions: subsets of ports defined by connectors
- **P**iorities: conflict resolution between interactions.

Real-Time extension of the BIP language and tools:

- *abstract* model: **timed** automata representing user requirements
- real-time execution on the target platform (actual execution times)
- static verification for known properties on execution times.

Methodology

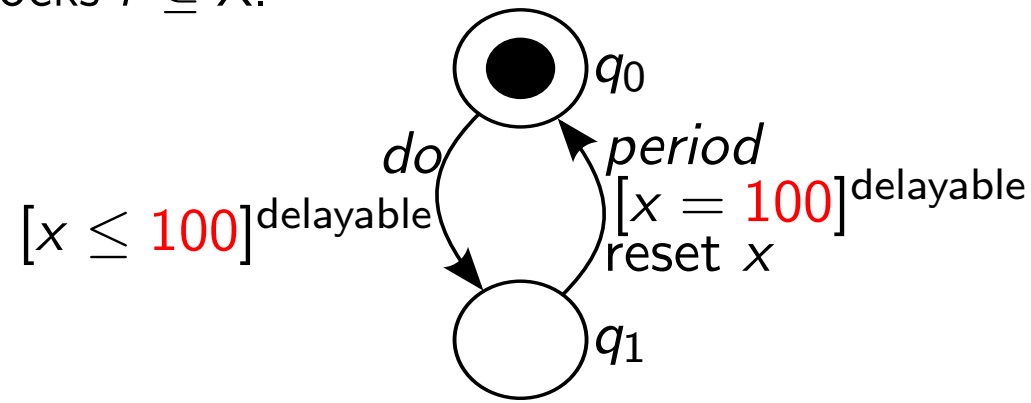
- From an (abstract) Real-Time BIP model:



Definition of Atomic Components (Timed Automata)

An atomic component is a timed automaton $M = (A, Q, X, \longrightarrow)$:

- set of actions A
- set of control locations Q, initial control location q_0
- set of clocks X
- labeled transitions $q \xrightarrow{a,g,r} q'$ between control locations:
  - **execute** an action $a \in A$
  - has a **timing constraint** g, i.e. a conjunction of simple constraints $x < K$, $x \leq K$, $x > K$, or $x \geq K$ ($K \in \mathbb{N}$), associated with an **urgency** type, i.e. lazy, delayable, or eager[1]
  - **reset** clocks $r \subseteq X$.

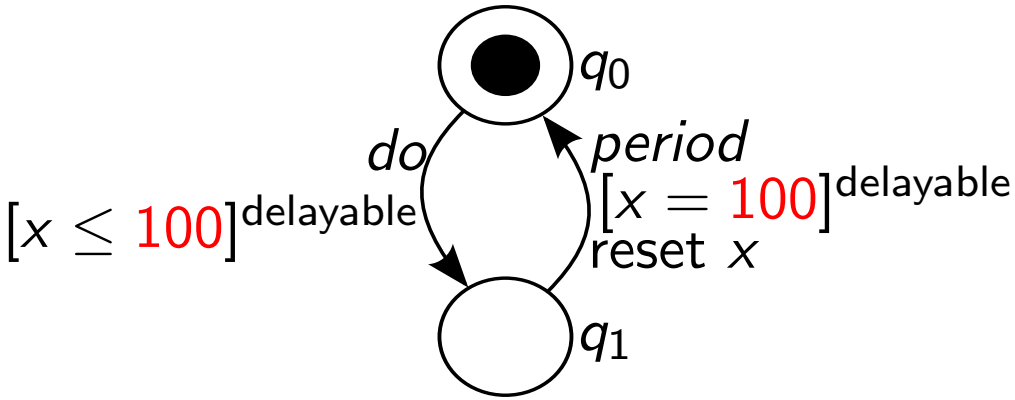


[1] instead of urgencies we may consider **time progress conditions** on control locations.

Semantics of Atomic Components (Timed Automata)

The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

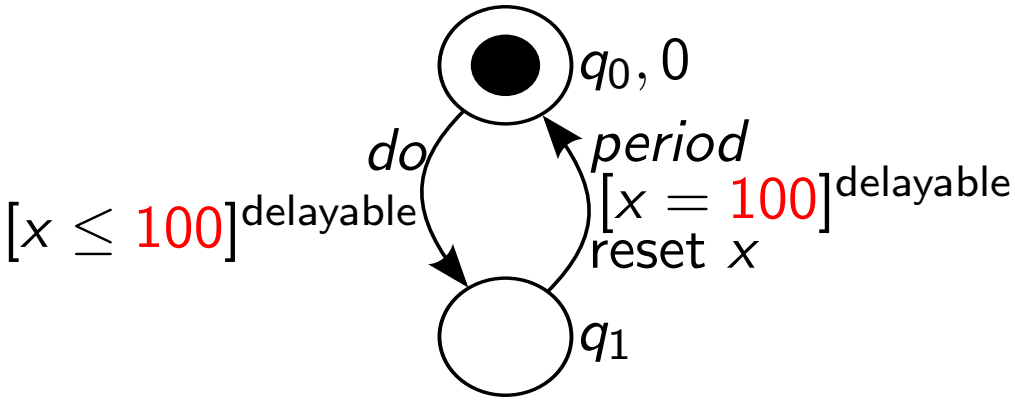
- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.

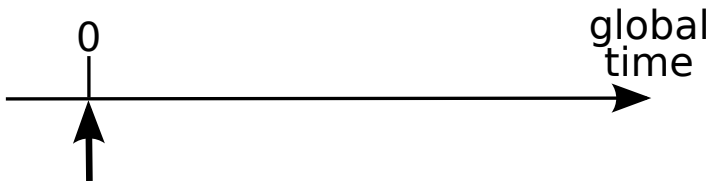


Semantics of Atomic Components (Timed Automata)

The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.

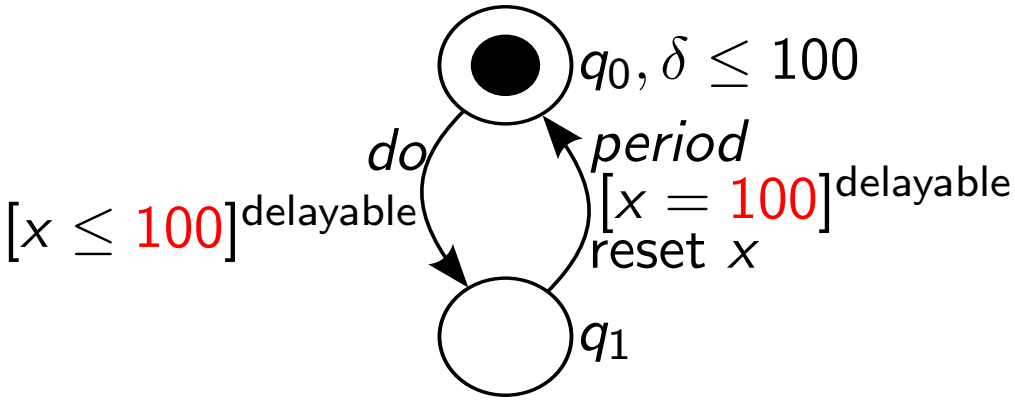


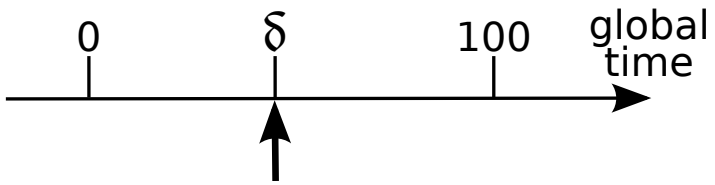


Semantics of Atomic Components (Timed Automata)

The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.

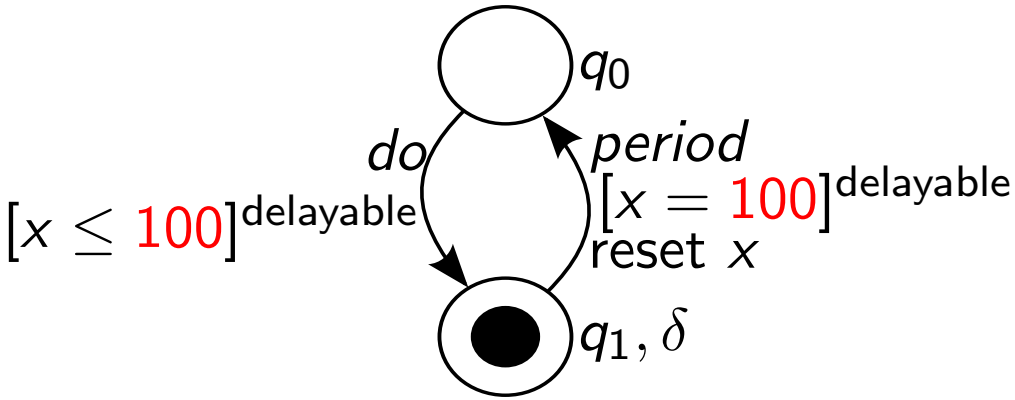


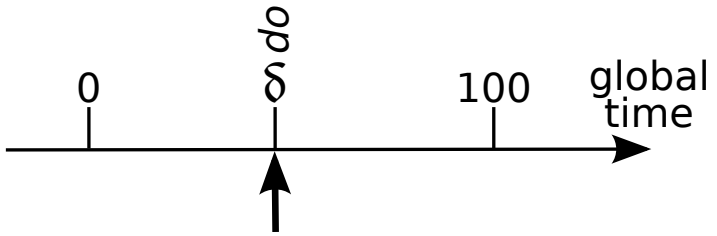


Semantics of Atomic Components (Timed Automata)

The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.

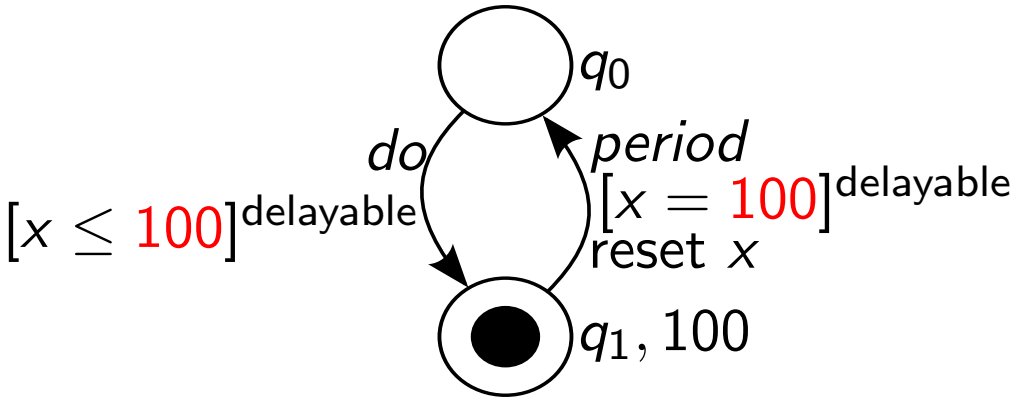


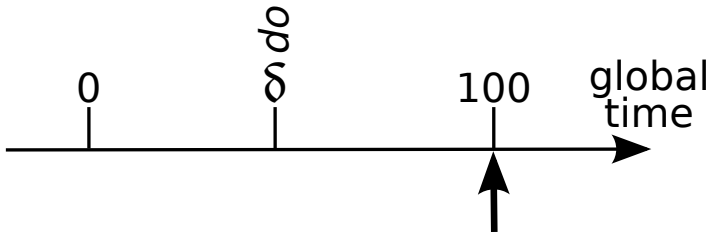


Semantics of Atomic Components (Timed Automata)

The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.

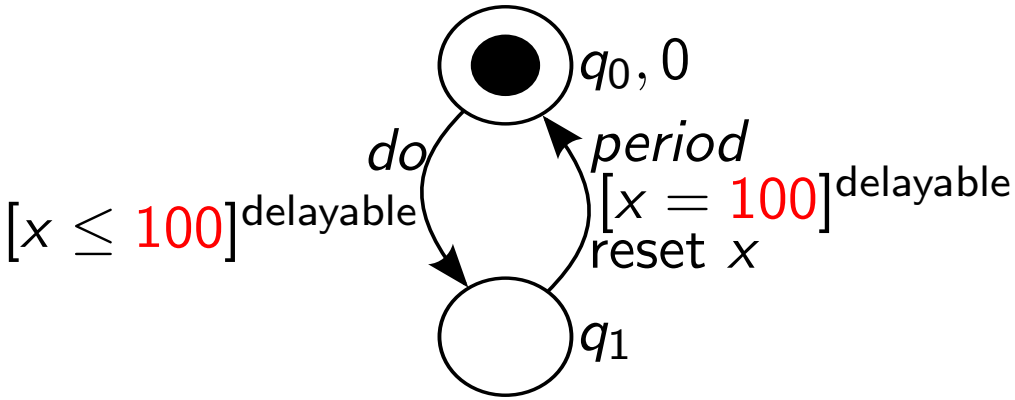


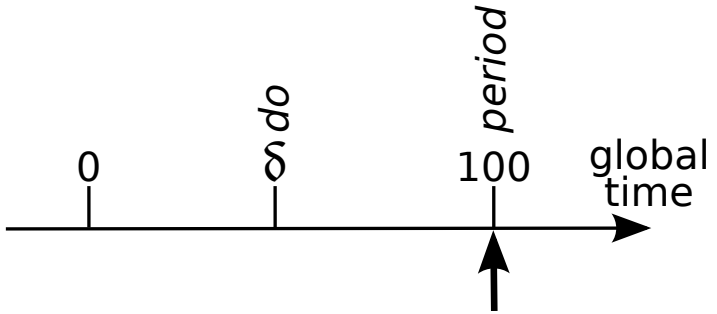


Semantics of Atomic Components (Timed Automata)

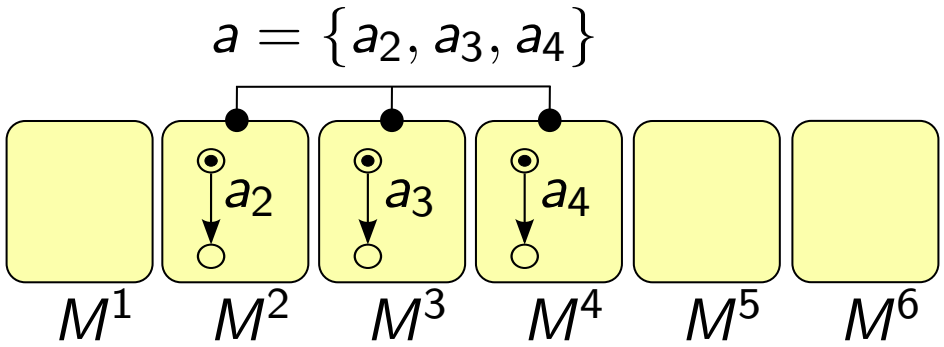
The semantics of a timed automaton $M = (A, Q, X, \longrightarrow)$ is an LTS:

- states: (q, v), $q \in Q$ and $v : X \to \mathbb{R}^+$
- transitions:
  - action $a \in A$: $(q, v) \xrightarrow{a} (q, v[r \to 0])$ if $g(v)$
  - time progress of $\delta \in \mathbb{R}^+$: $(q, v) \xrightarrow{\delta} (q, v + \delta)$
    if $\forall \delta' \in [0, \delta[\not\exists q \xrightarrow{a,g,r} q'$ s.t. $urg[g](v + \delta')$ where
    $urg[g] = false$ for lazy transitions
    $urg[g] =\downarrow g$ for delayable transitions
    $urg[g] = g$ for eager transitions.





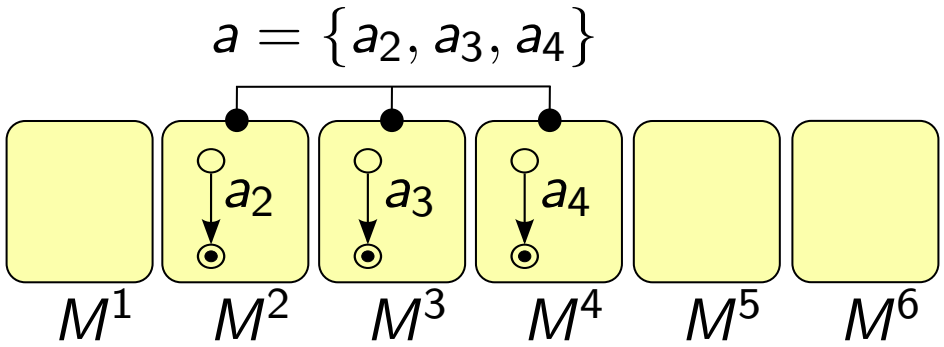
Composition of Components: Interactions



Given n timed automata $M^i = (\mathrm{A}^i, \mathrm{Q}^i, \mathrm{X}^i, \longrightarrow^i)$, $1 \leq i \leq n$:

- an interaction a is a subset of actions $a \subseteq A = \bigcup_i A^i$ s.t. $a = \{a_i\}_{i \in I}$, $I \subseteq \{1, \ldots, n\}$
- the composition of components M^i for a set of interactions γ is a timed automata $\gamma(M^1, \ldots, M^n) = (\mathrm{A}, \mathrm{Q}, \mathrm{X}, \longrightarrow)$ where:
  - $\mathrm{Q} = \mathrm{Q}^1 \times \ldots \times \mathrm{Q}^n$
  - $\mathrm{X} = \bigcup_i \mathrm{X}^i$
  - $(q_1, \ldots, q_n) \xrightarrow{a, g_a, r} (q'_1, \ldots, q'_n)$ if $a = \{a_i\}_{i \in I} \in \gamma$ and:
    - $q_i \xrightarrow{a_i, g_i, r_i} q'_i$ for all $i \in I$, and $q'_i = q_i$ for $i \notin I$
    - $g_a = \bigwedge_{i \in I} g_i$ (for urgencies we take the max, lazy $<$ delayable $<$ eager)
    - $r = \bigcup_{i \in I} r_i$

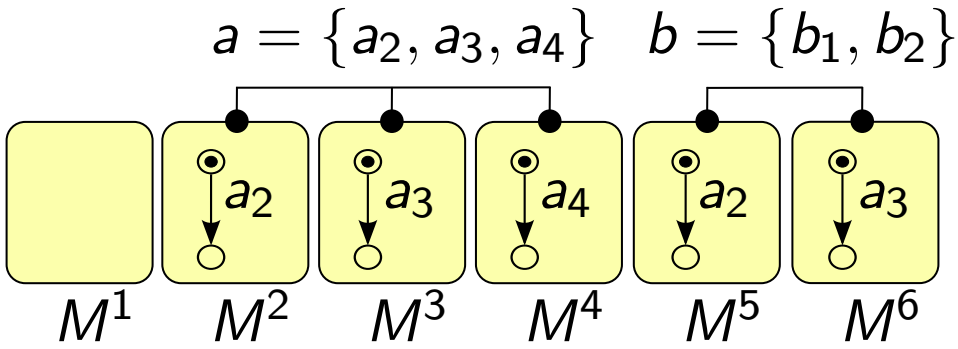
Composition of Components: Interactions



Given n timed automata $M^i = (\mathsf{A}^i, \mathsf{Q}^i, \mathsf{X}^i, \longrightarrow^i)$, $1 \leq i \leq n$:

- an interaction a is a subset of actions $a \subseteq A = \bigcup_i A^i$ s.t. $a = \{a_i\}_{i \in I}$, $I \subseteq \{1, \ldots, n\}$
- the composition of components M^i for a set of interactions γ is a timed automata $\gamma(M^1, \ldots, M^n) = (\mathsf{A}, \mathsf{Q}, \mathsf{X}, \longrightarrow)$ where:
  - $\mathsf{Q} = \mathsf{Q}^1 \times \ldots \times \mathsf{Q}^n$
  - $\mathsf{X} = \bigcup_i \mathsf{X}^i$
  - $(q_1, \ldots, q_n) \xrightarrow{a, g_a, r} (q'_1, \ldots, q'_n)$ if $a = \{a_i\}_{i \in I} \in \gamma$ and:
    - $q_i \xrightarrow{a_i, g_i, r_i} q'_i$ for all $i \in I$, and $q'_i = q_i$ for $i \notin I$
    - $g_a = \bigwedge_{i \in I} g_i$ (for urgencies we take the max, lazy $<$ delayable $<$ eager)
    - $r = \bigcup_{i \in I} r_i$

Composition of Components: Priorities

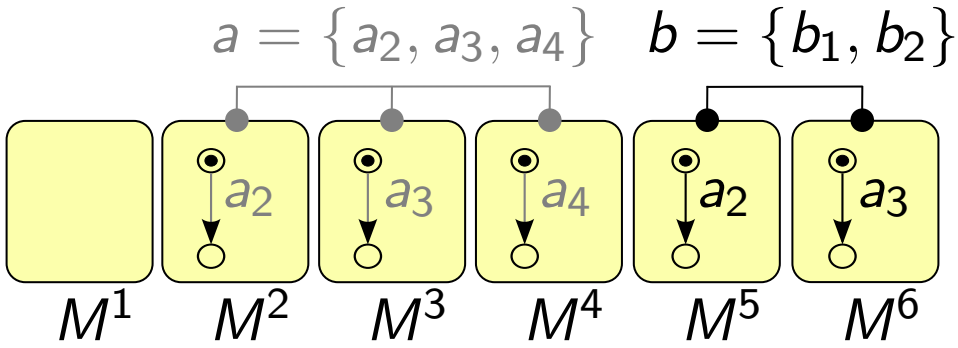


- priorites π is a partial order on interactions restricting their execution, i.e. $\pi\gamma(M^1, \ldots, M^n)$ is obtained from $\gamma(M^1, \ldots, M^n)$ using:

$$g'_a = g_a \wedge \neg \bigwedge_{a<b} g_b$$

Composition of Components: Priorities

$$a < b$$



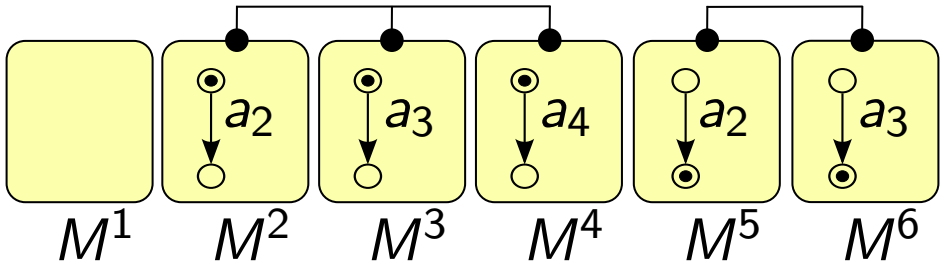
- priorites π is a partial order on interactions restricting their execution, i.e. $\pi\gamma(M^1, \ldots, M^n)$ is obtained from $\gamma(M^1, \ldots, M^n)$ using:

$$g'_a = g_a \wedge \neg \bigwedge_{a<b} g_b$$

Composition of Components: Priorities

$$a < b$$

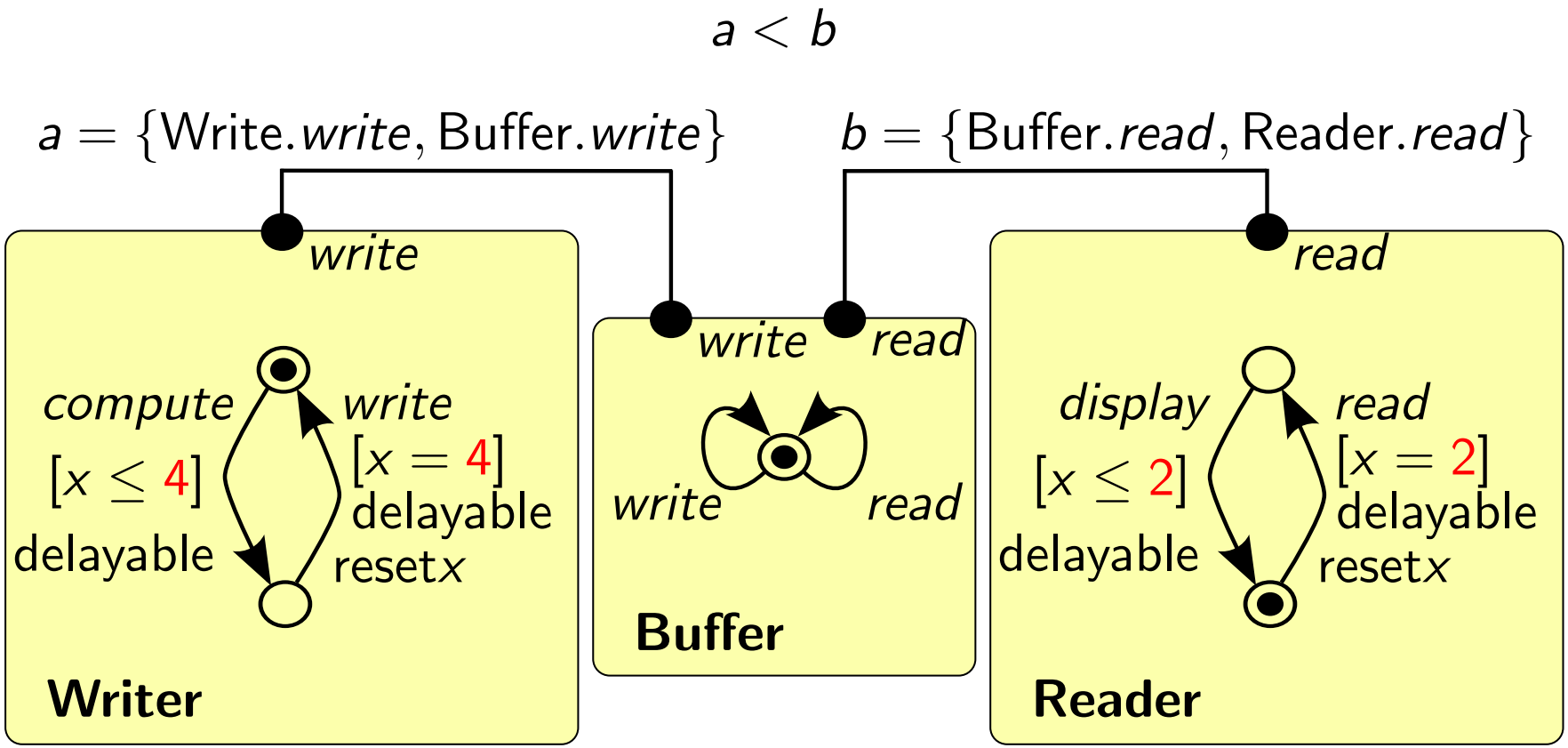
$$a = \{a_2, a_3, a_4\} \quad b = \{b_1, b_2\}$$



- priorites π is a partial order on interactions restricting their execution, i.e. $\pi\gamma(M^1, \ldots, M^n)$ is obtained from $\gamma(M^1, \ldots, M^n)$ using:

$$g'_a = g_a \wedge \neg \bigwedge_{a<b} g_b$$

Example


$a < b$
$a = \{\text{Write}.write, \text{Buffer}.write\}$
$b = \{\text{Buffer}.read, \text{Reader}.read\}$
write
compute
$[x \leq 4]$
delayable
write
$[x = 4]$
delayable
reset$x$
Writer
write
read
write
read
Buffer
read
display
$[x \leq 2]$
delayable
read
$[x = 2]$
delayable
reset$x$
Reader

From Abstract to Physical Model

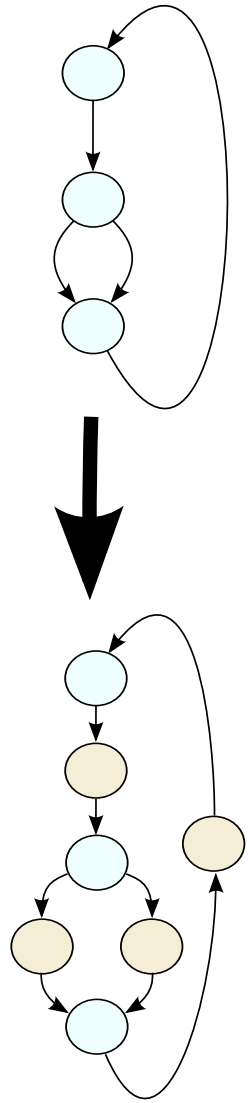
timeless execution of actions +
begin & end of executions coincide +

Abstract Model: timed automata

platform properties
execution times +

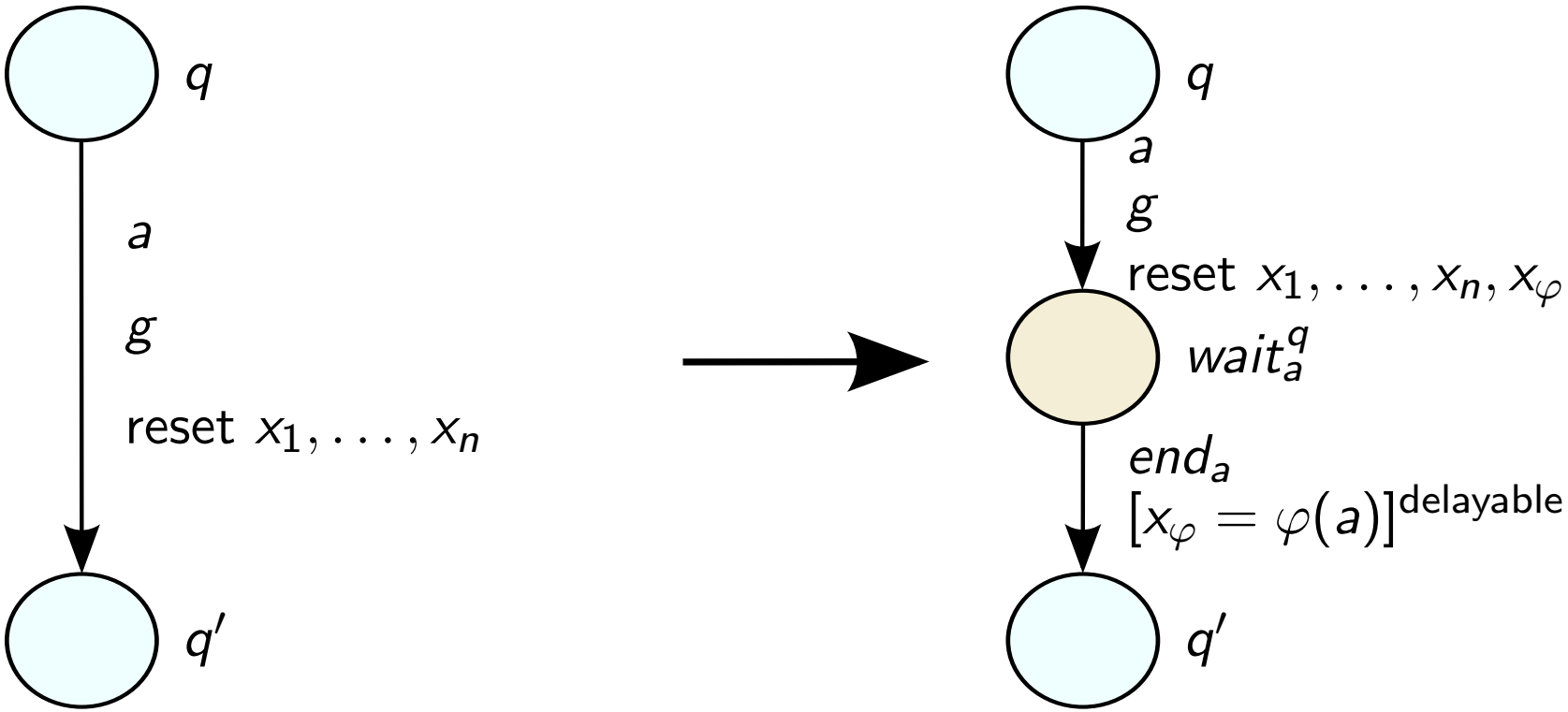
modified abstract model +
transitions split into begin and end +
waiting times corresponding to +
exec. times using an additional clock

Physical Model: timed automata



From Abstract to Physical Model

- Given execution times $\varphi : A \to \mathbb{R}^+$:



- Abstract Model M:
  - timeless execution
  - $M = M_0$.
- Physical Model M_φ:
  - execution time of a is $\varphi(a)$
  - g applies to the beginning of a.

Physical Model vs Abstract Model

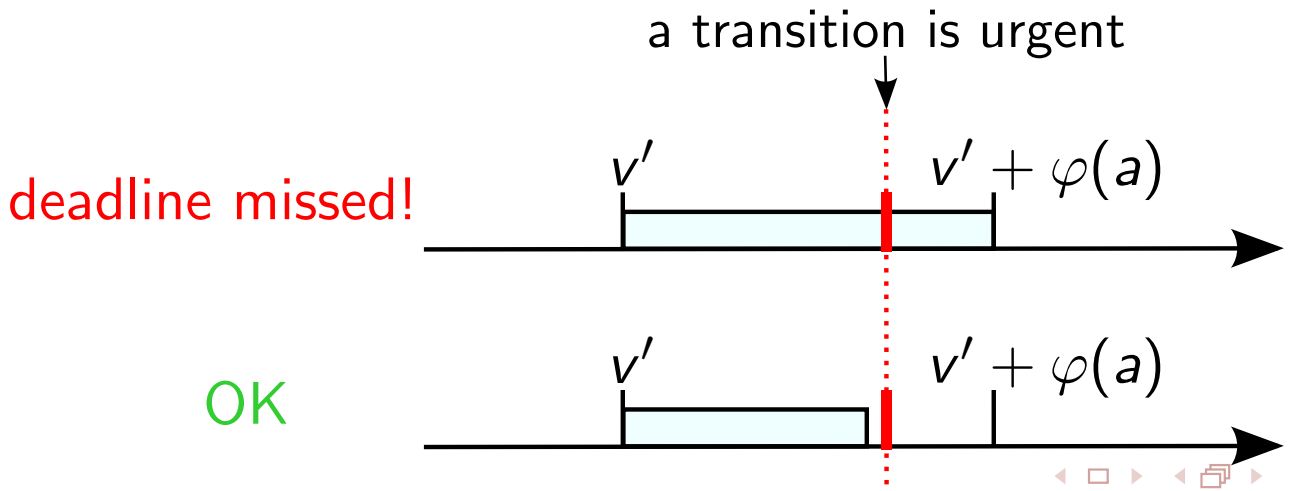
- In physical model M_φ each execution of a is immediately followed by a waiting time of $\varphi(a)$:
  - in M_φ: $\quad (q, v) \xrightarrow{a} (wait_a^q, v') \xrightarrow{\varphi(a)} (q', v' + \varphi(a))$
  - in M: $\quad (q, v) \xrightarrow{a} (q', \quad v')$

Physical Model vs Abstract Model

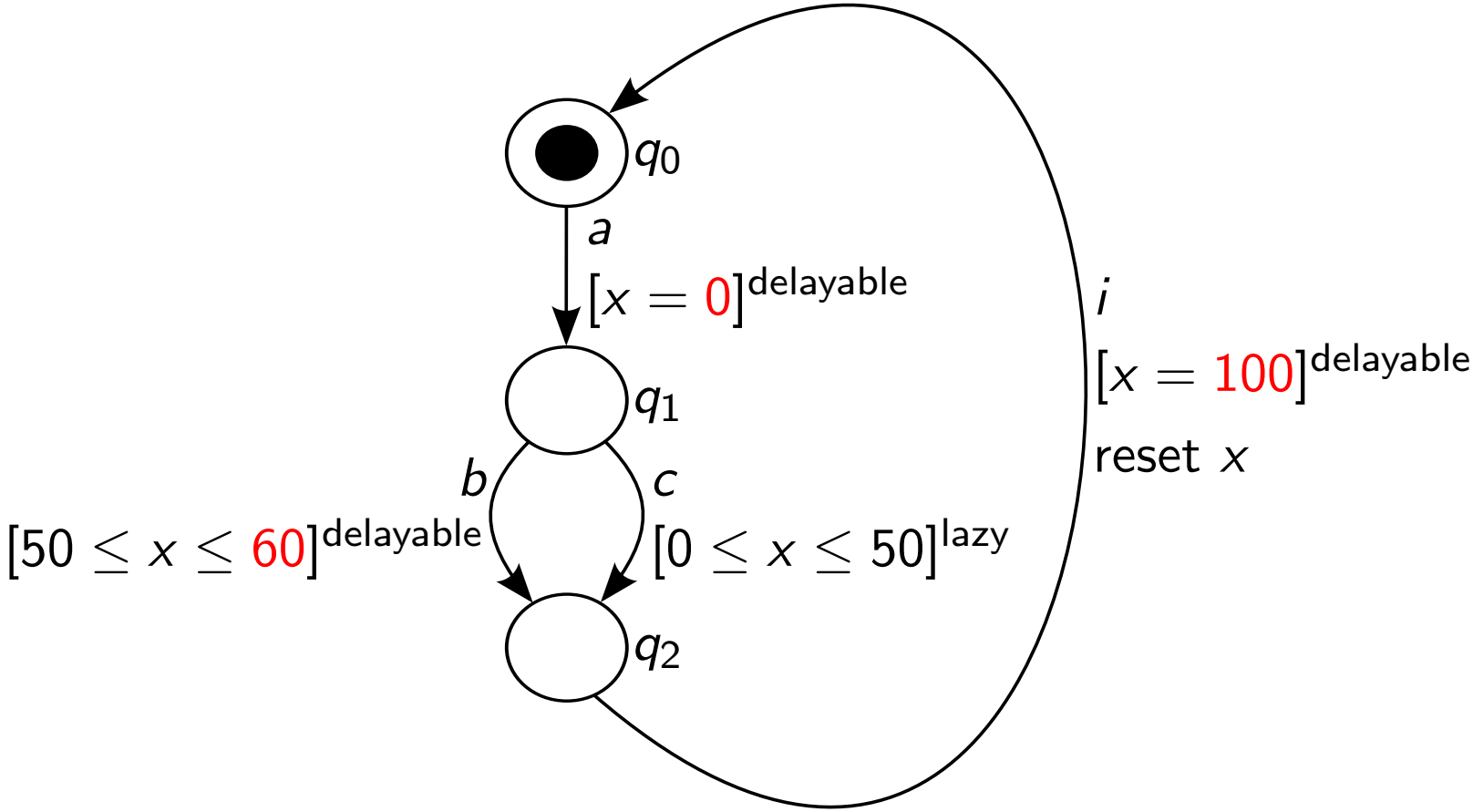
- In physical model M_φ each execution of a is immediately followed by a waiting time of $\varphi(a)$:
  - in M_φ: $(q, v) \xrightarrow{a} (wait_a^q, v') \xrightarrow{\varphi(a)} (q', v' + \varphi(a))$
  - in M: $(q, v) \xrightarrow{a} (q', \quad v') \xrightarrow{\delta} (q', v' + \delta)$

$\exists \delta = \varphi(a)?$

- Is there in M an execution sequence equivalent to that of M_φ, i.e., no transition is urgent in $[v', v' + \varphi(a)[$.



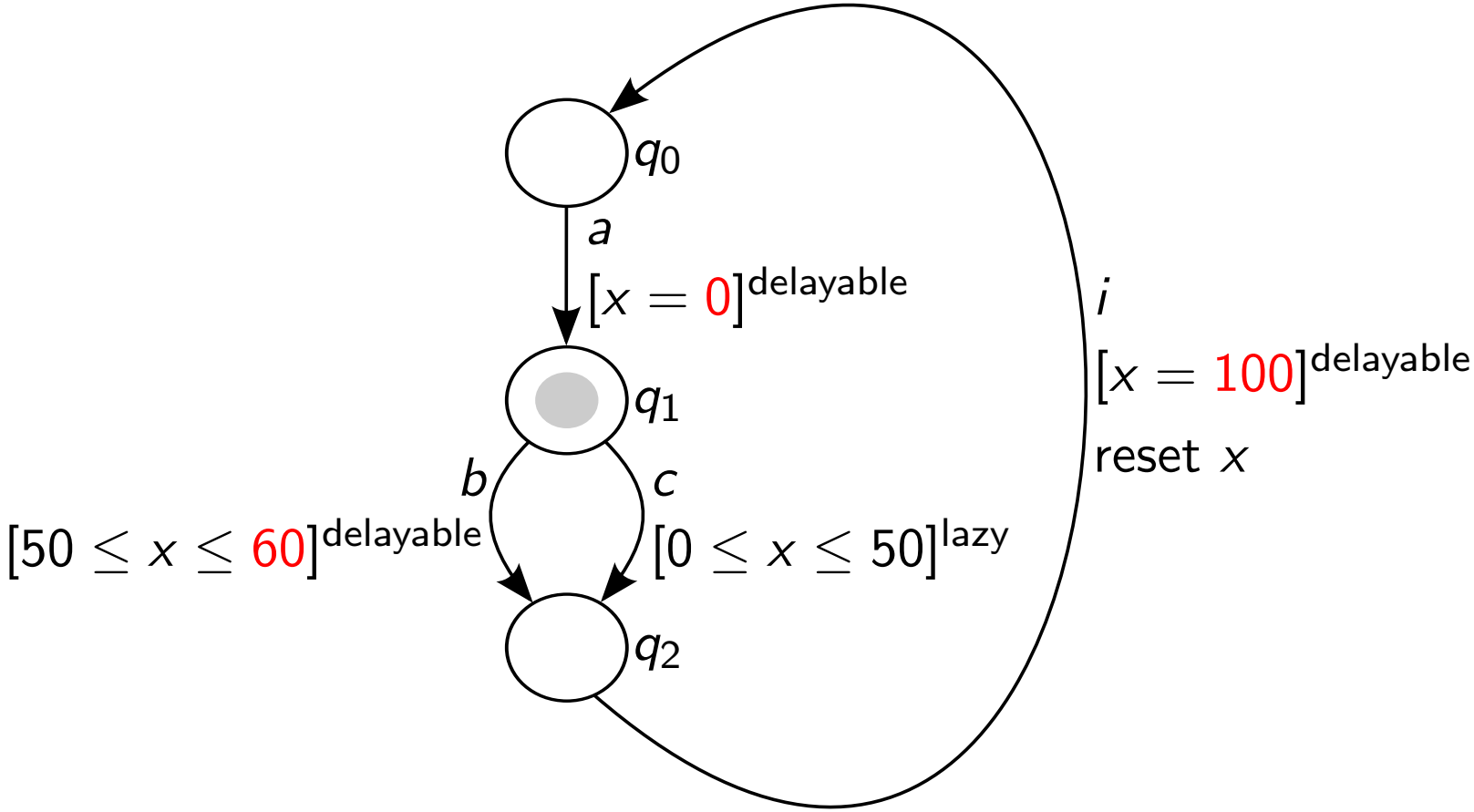
Example of Deadline Miss



- In M_φ such that $\varphi(a) = 70$, we have:

$(q_0, 0)$

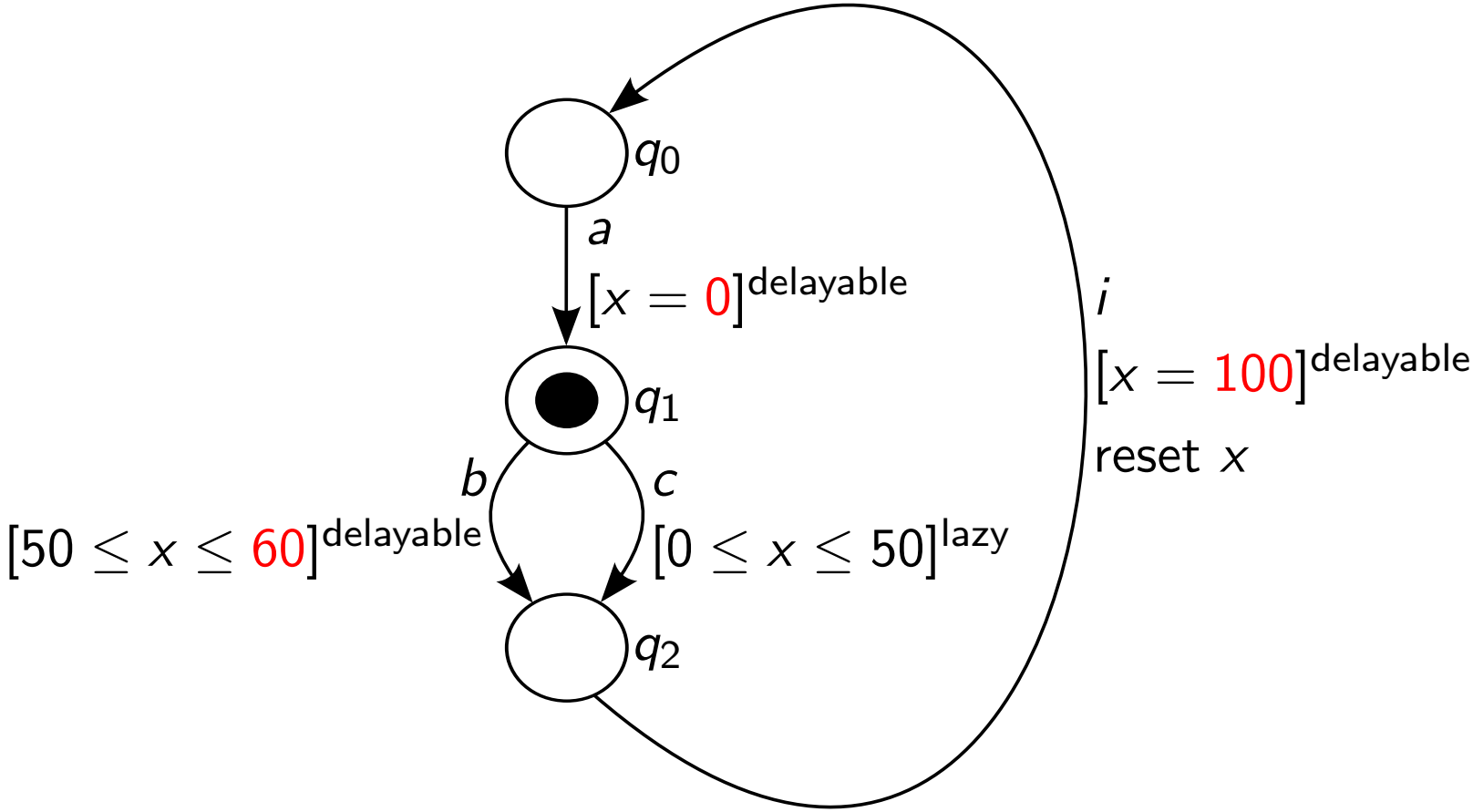
Example of Deadline Miss



- In M_φ such that $\varphi(a) = 70$, we have:

$$(q_0, 0) \xrightarrow{a} (q_1, 0)$$

Example of Deadline Miss



- In M_φ such that $\varphi(a) = 70$, we have:

$$(q_0, 0) \xrightarrow{a} (q_1, 0) \underbrace{\xrightarrow{\varphi(a)=70} \overbrace{(q_1, 70)}^{\text{deadlock}}}_{\text{deadline 60 missed!}}$$

A Notion of Time-Safety

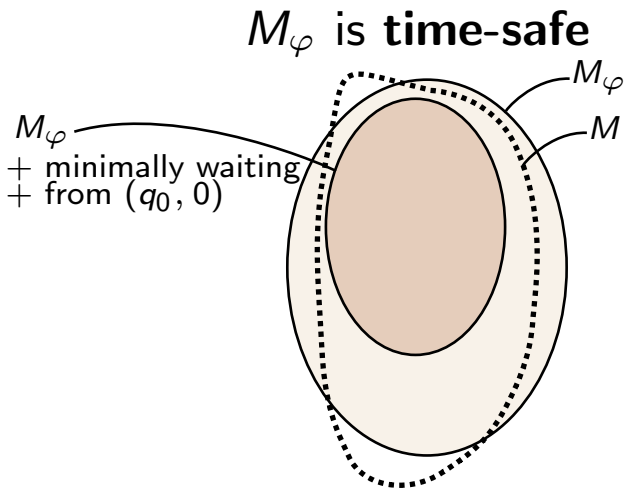
- We are only interested in minimally waiting schedulers, i.e.:

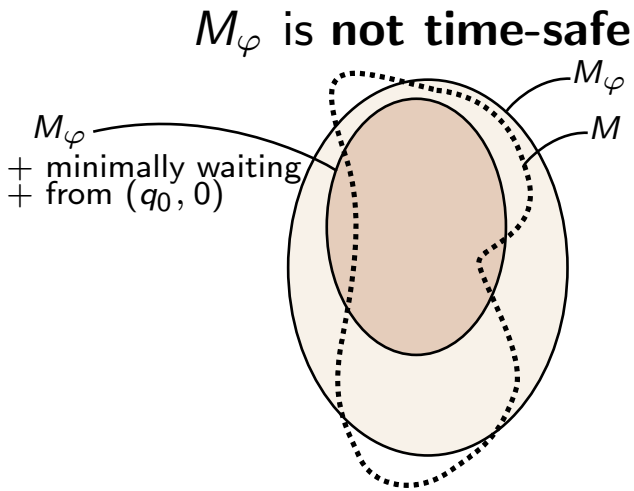
  we do not consider $(q, v) \xrightarrow{\delta} (q, v+\delta) \xrightarrow{a}$

  if $\underline{\exists \delta' < \delta}$ $(q, v) \xrightarrow{\delta'} (q, v+\delta') \xrightarrow{a}$

Definition (Time-Safety)

A *physical* model M_φ is time-safe if its execution sequences are also execution sequences of the *abstract* model M, i.e. **no deadline miss**.





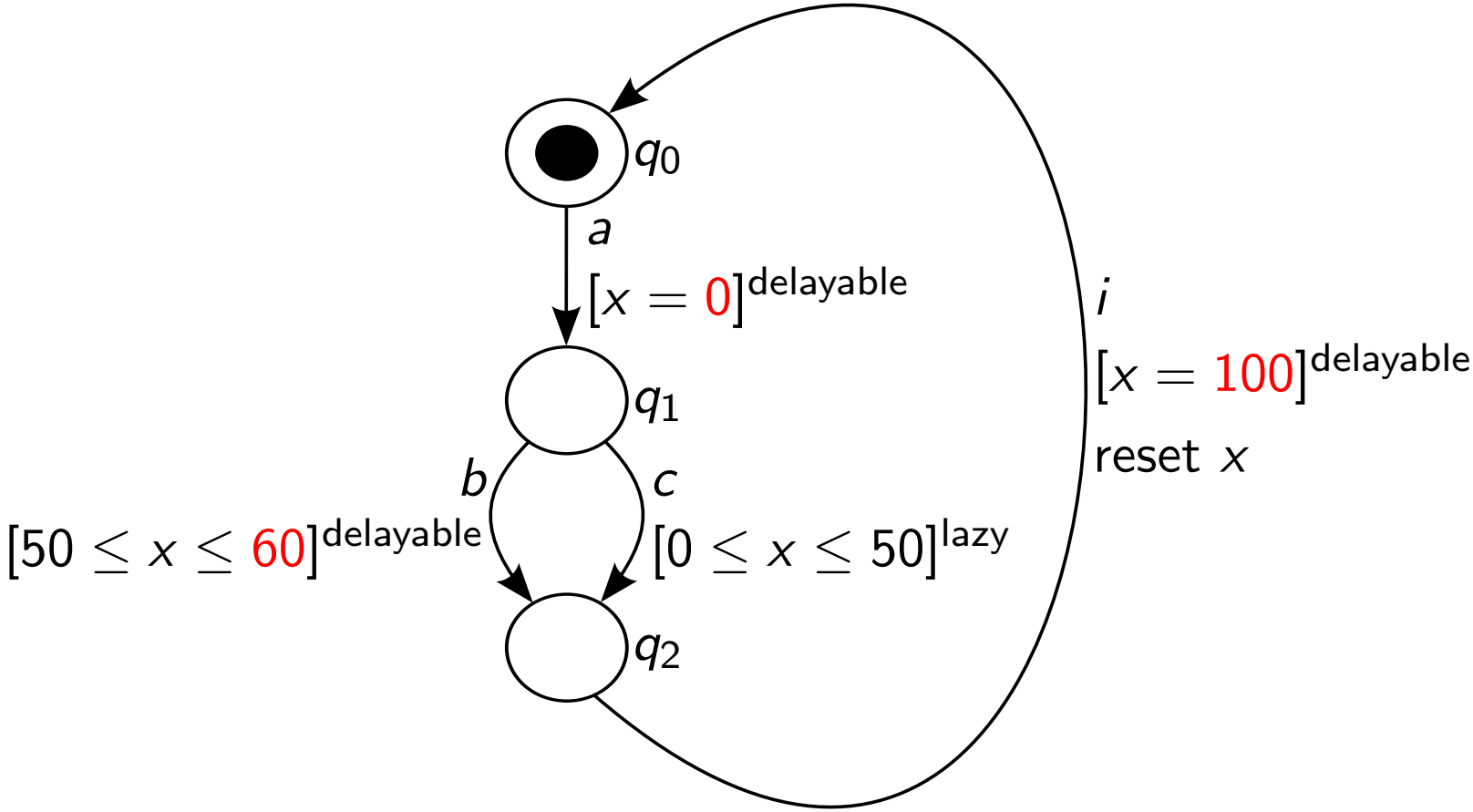
A Notion of Time-Robustness

Definition (Time-Robustness)

- A time-safe *physical* model M_φ is time-robust if reducing execution times preserves time-safety, i.e. for all $\varphi' \leq \varphi$, $M_{\varphi'}$ is time-safe.
- An *abstract* model is time-robust if all its time-safe *physical* models are time-robust.

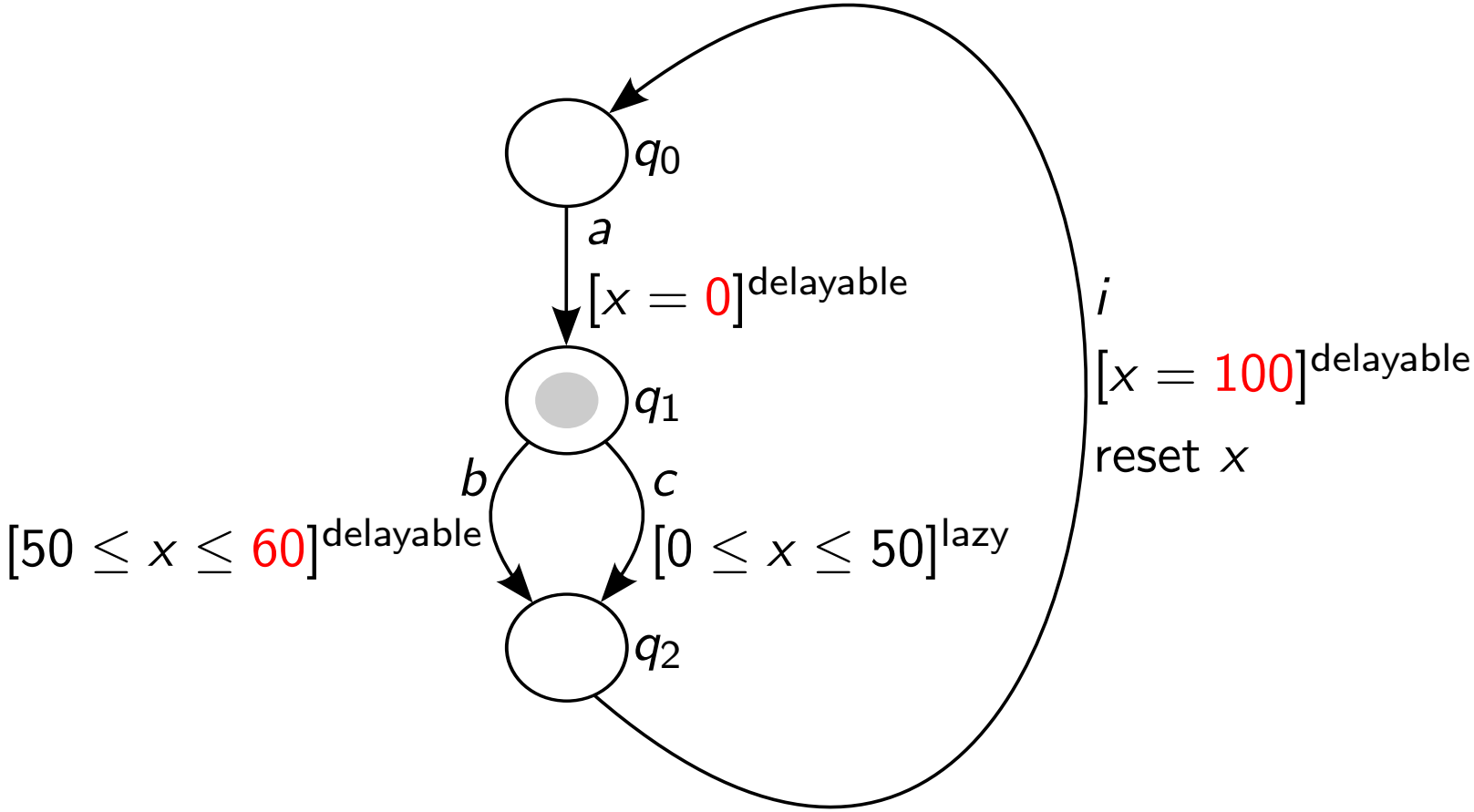
- Robustness allows checking time-safety only for WCETs φ^{wc}.
  In case of robustness, time-safety for any execution times φ is obtained by the fact that $\varphi \leq \varphi^{wc}$.

Non Time-Robust Example



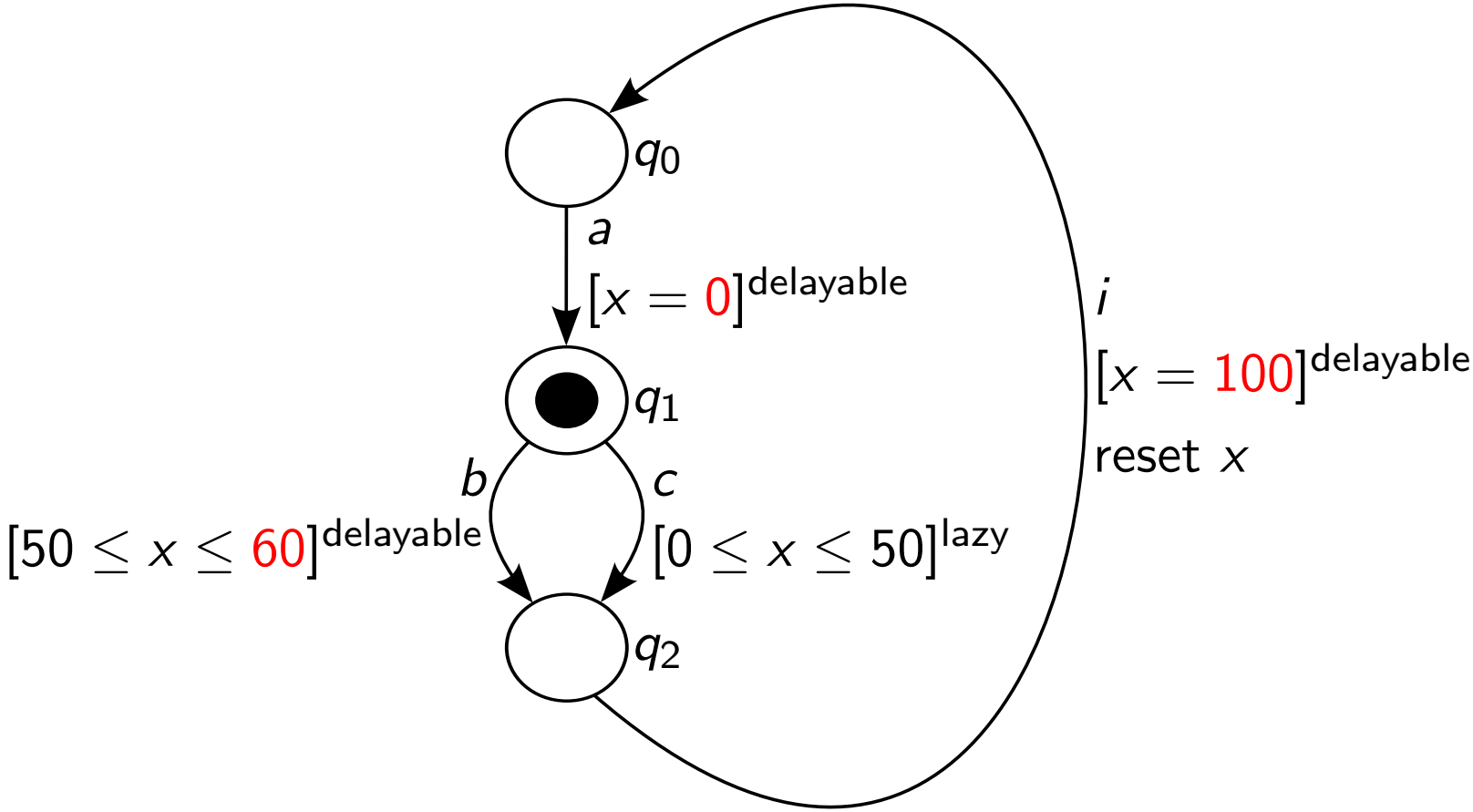
- $(q_0, 0)$

Non Time-Robust Example



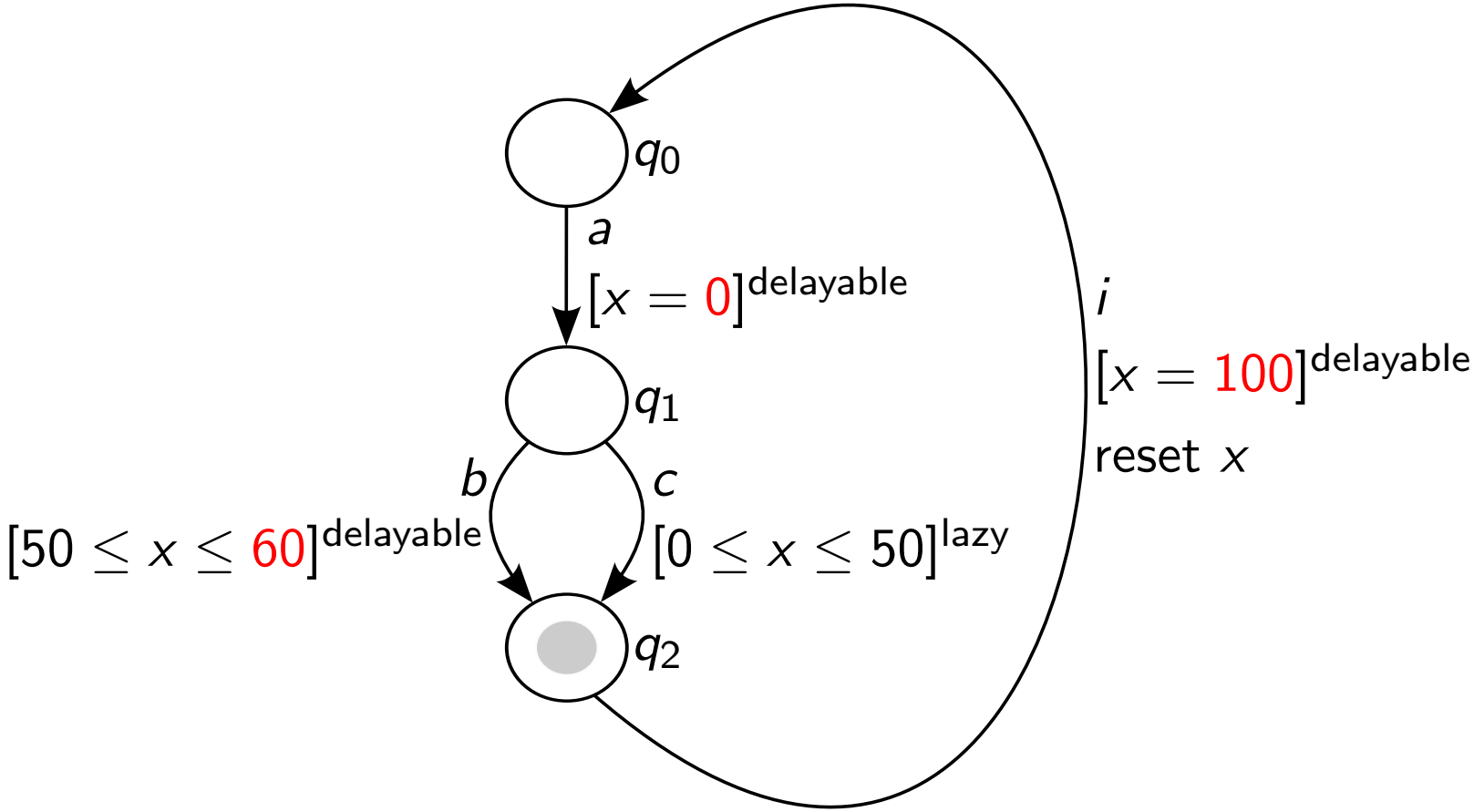
- $(q_0, 0) \xrightarrow{a} (q_1, 0)$

Non Time-Robust Example



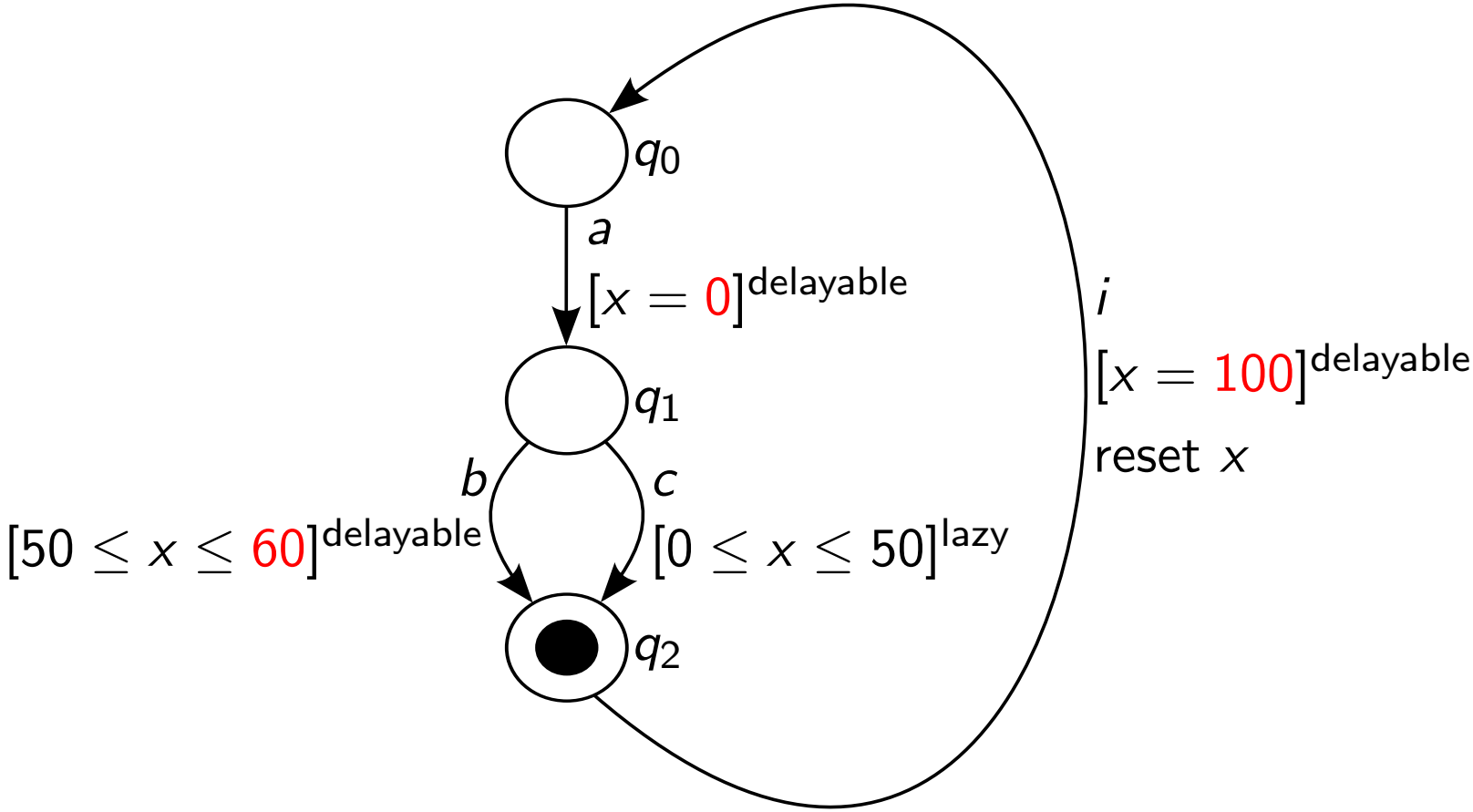
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60)$

Non Time-Robust Example



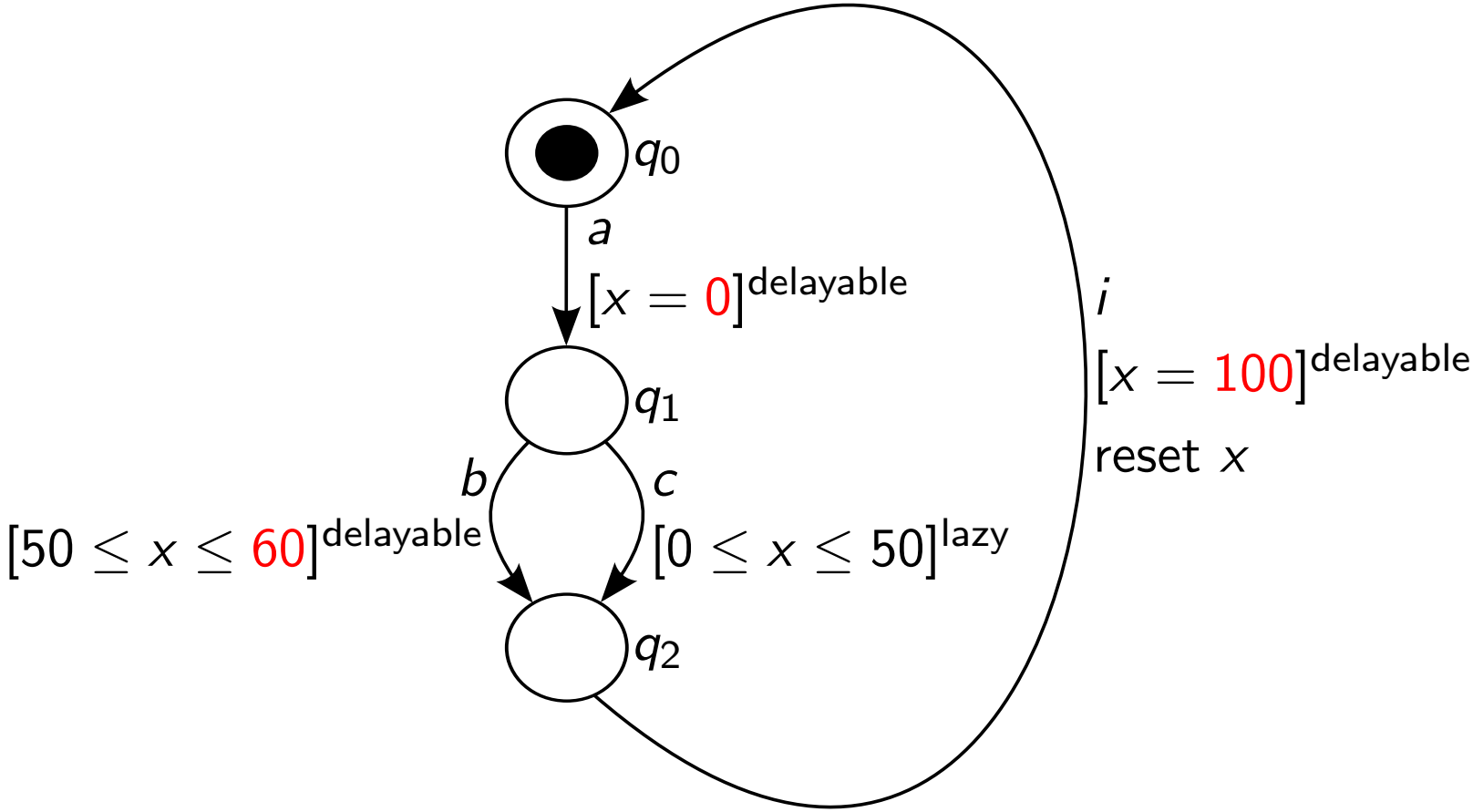
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60) \xrightarrow{b} (q_2, 60)$

Non Time-Robust Example



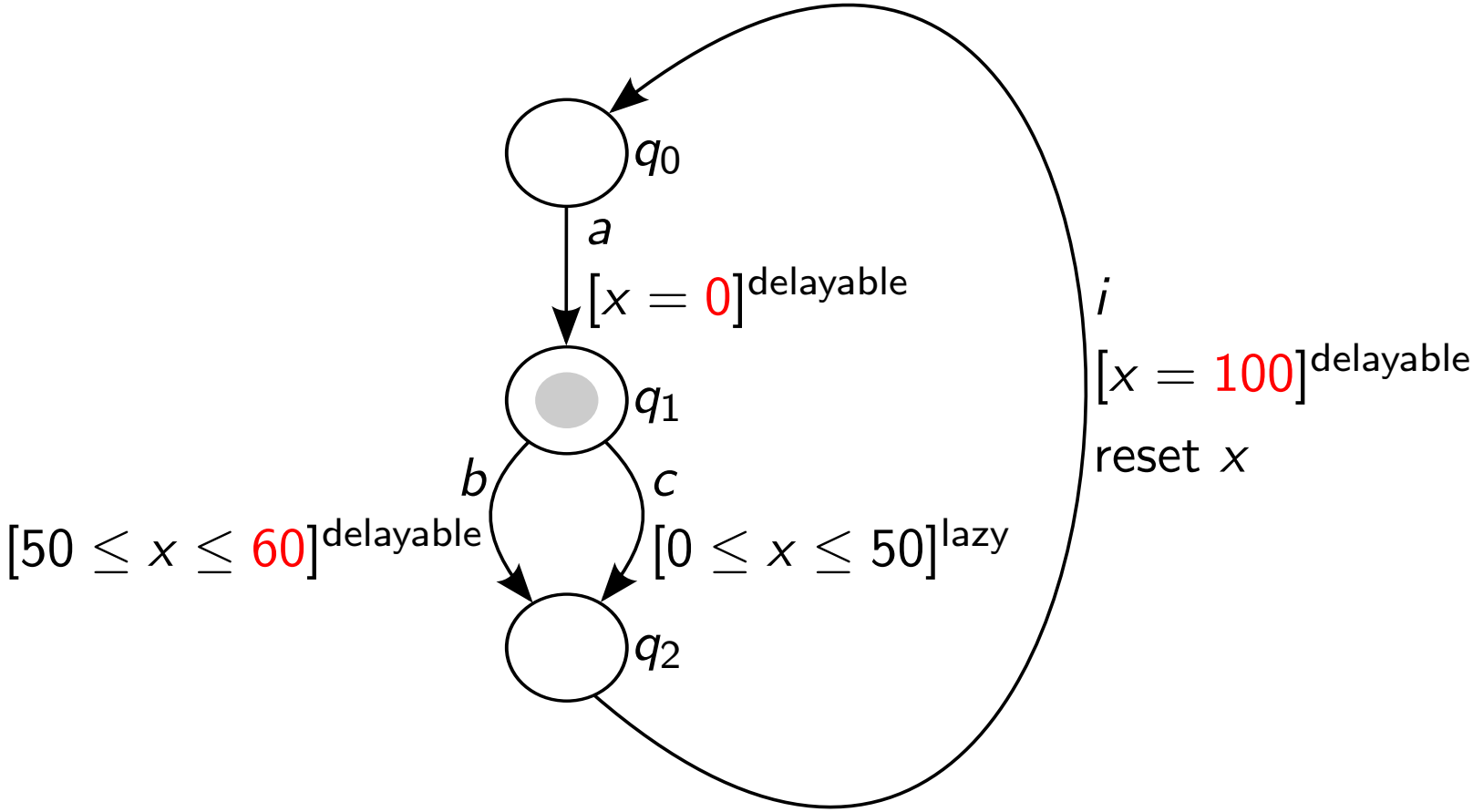
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60) \xrightarrow{b} (q_2, 60) \xrightarrow{\varphi(b)=40} (q_2, 100)$

Non Time-Robust Example



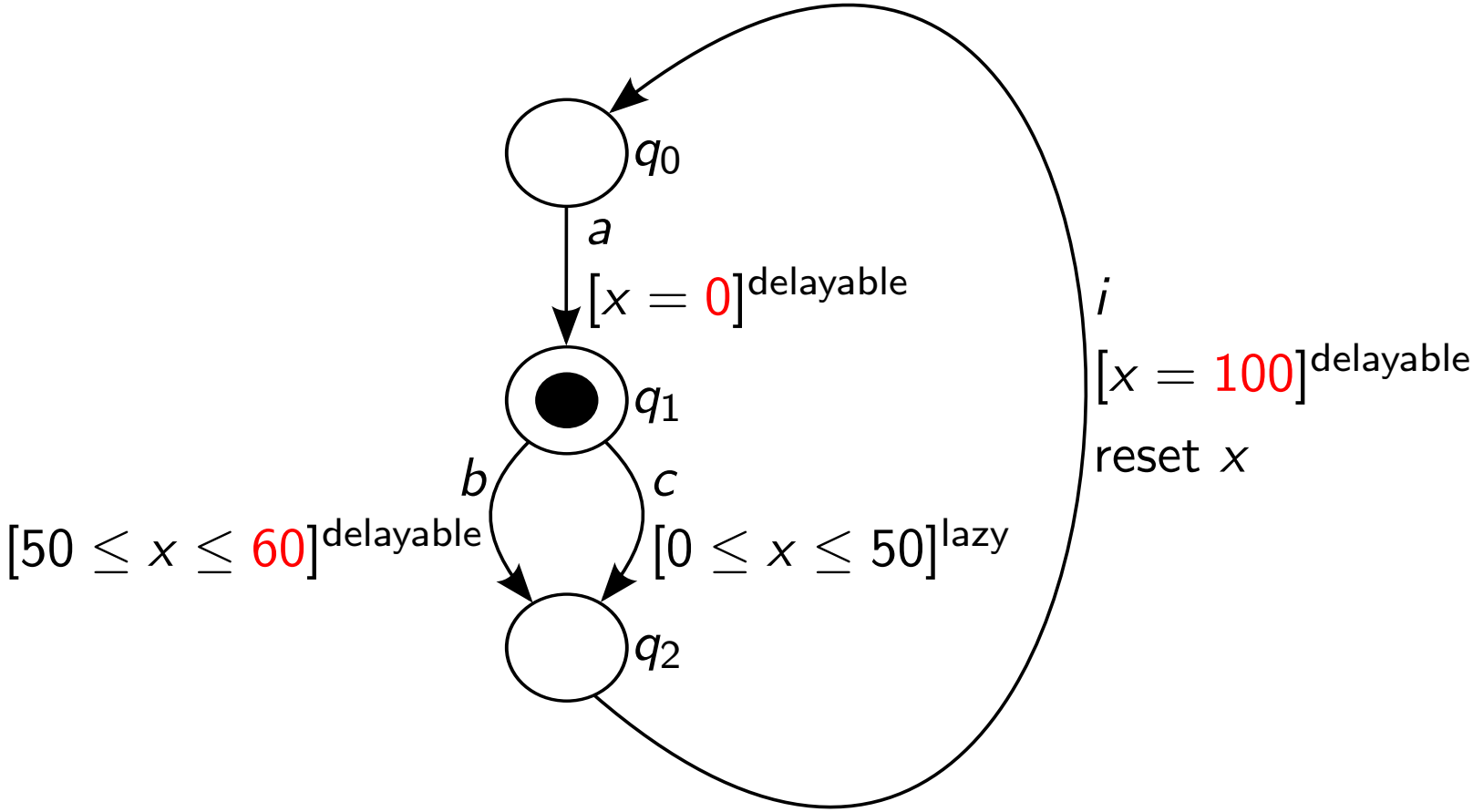
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60) \xrightarrow{b} (q_2, 60) \xrightarrow{\varphi(b)=40} (q_2, 100) \xrightarrow{i} (q_0, 0)$

Non Time-Robust Example



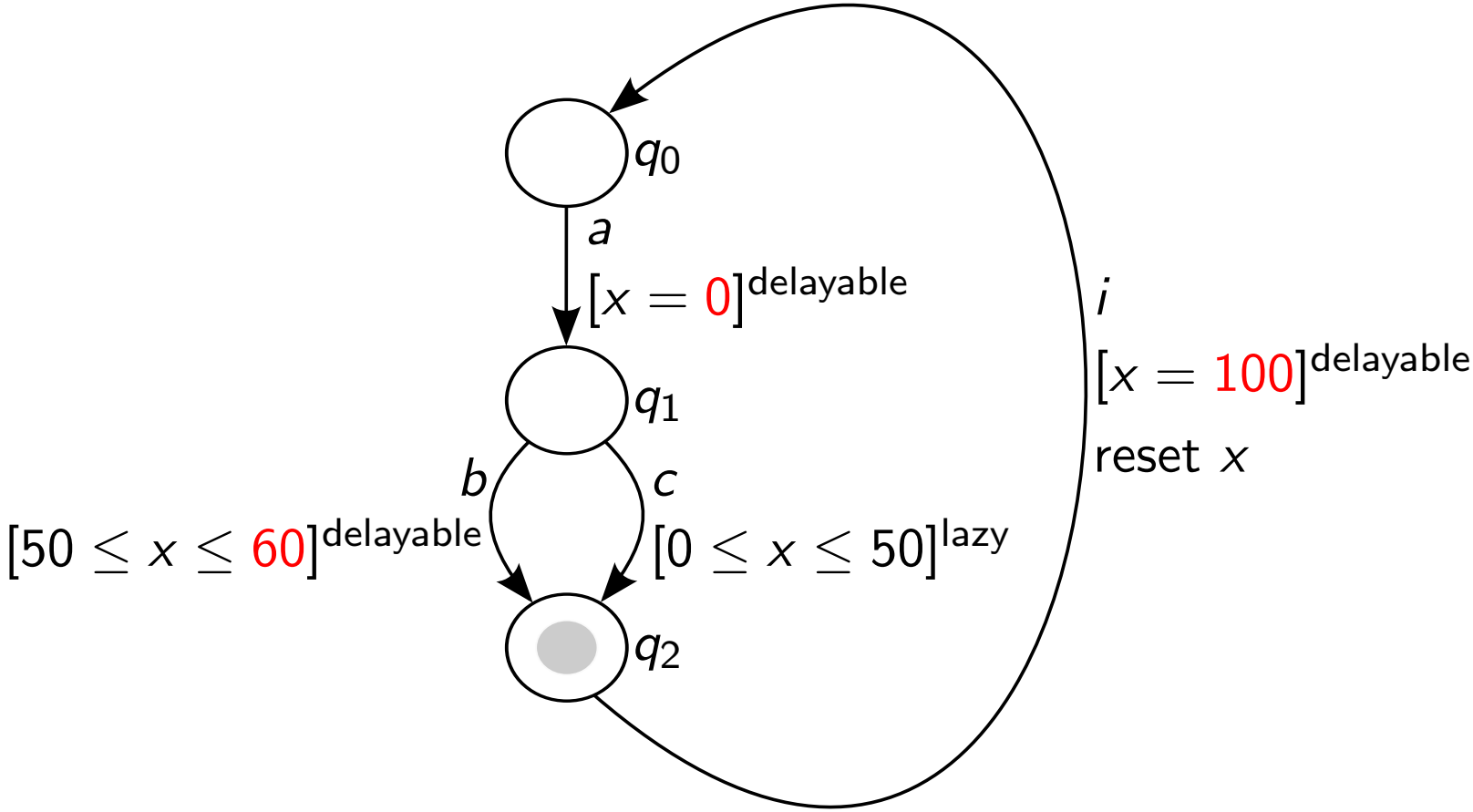
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \overset{\varphi(a)=60}{\longrightarrow} (q_1, 60) \xrightarrow{b} (q_2, 60) \overset{\varphi(b)=40}{\longrightarrow} (q_2, 100) \xrightarrow{i} (q_0, 0)$
- $(q_0, 0) \xrightarrow{a} (q_1, 0)$

Non Time-Robust Example



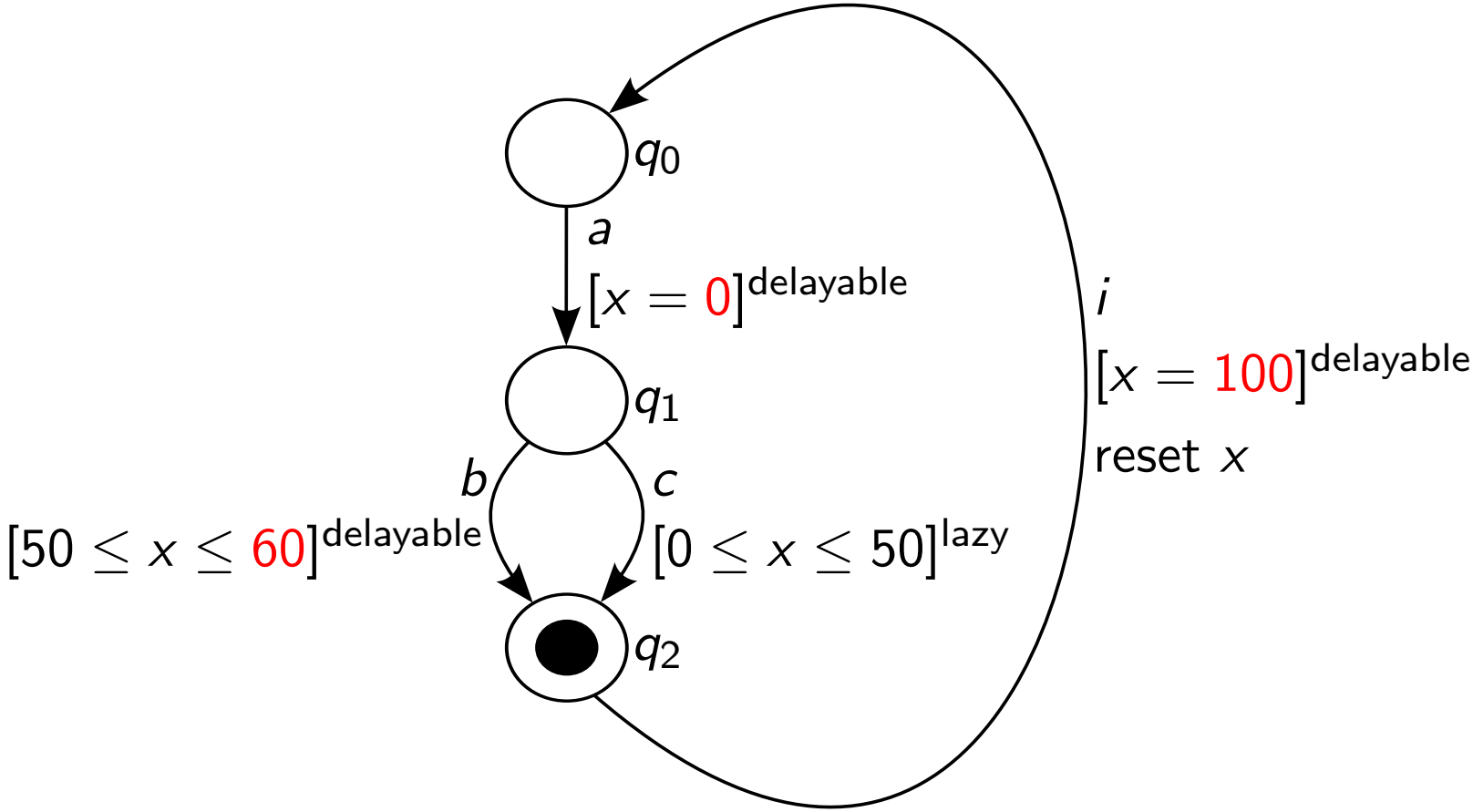
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60) \xrightarrow{b} (q_2, 60) \xrightarrow{\varphi(b)=40} (q_2, 100) \xrightarrow{i} (q_0, 0)$
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=50} (q_1, 50)$

Non Time-Robust Example



- $(q_0, 0) \xrightarrow{a} (q_1, 0) \overset{\varphi(a)=60}{\longrightarrow} (q_1, 60) \xrightarrow{b} (q_2, 60) \overset{\varphi(b)=40}{\longrightarrow} (q_2, 100) \xrightarrow{i} (q_0, 0)$
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \overset{\varphi(a)=50}{\longrightarrow} (q_1, 50) \xrightarrow{c} (q_2, 50)$

Non Time-Robust Example



- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=60} (q_1, 60) \xrightarrow{b} (q_2, 60) \xrightarrow{\varphi(b)=40} (q_2, 100) \xrightarrow{i} (q_0, 0)$
- $(q_0, 0) \xrightarrow{a} (q_1, 0) \xrightarrow{\varphi(a)=50} (q_1, 50) \xrightarrow{c} (q_2, 50) \underbrace{\xrightarrow{\varphi(c)=60} \overbrace{(q_2, 110)}^{\text{deadlock}}}_{\text{deadline 100 missed!}}$

Achieving Time-Robustness by Time-Determinism

Definition (Time-Determinism)

An abstract model is time-deterministic if all its guards are non-lazy equalities.

Proposition

Time-deterministic abstract models are *time-robust*.

$\rightarrow$ For time-deterministic models, it is sufficient to check time-safety for WCETs.

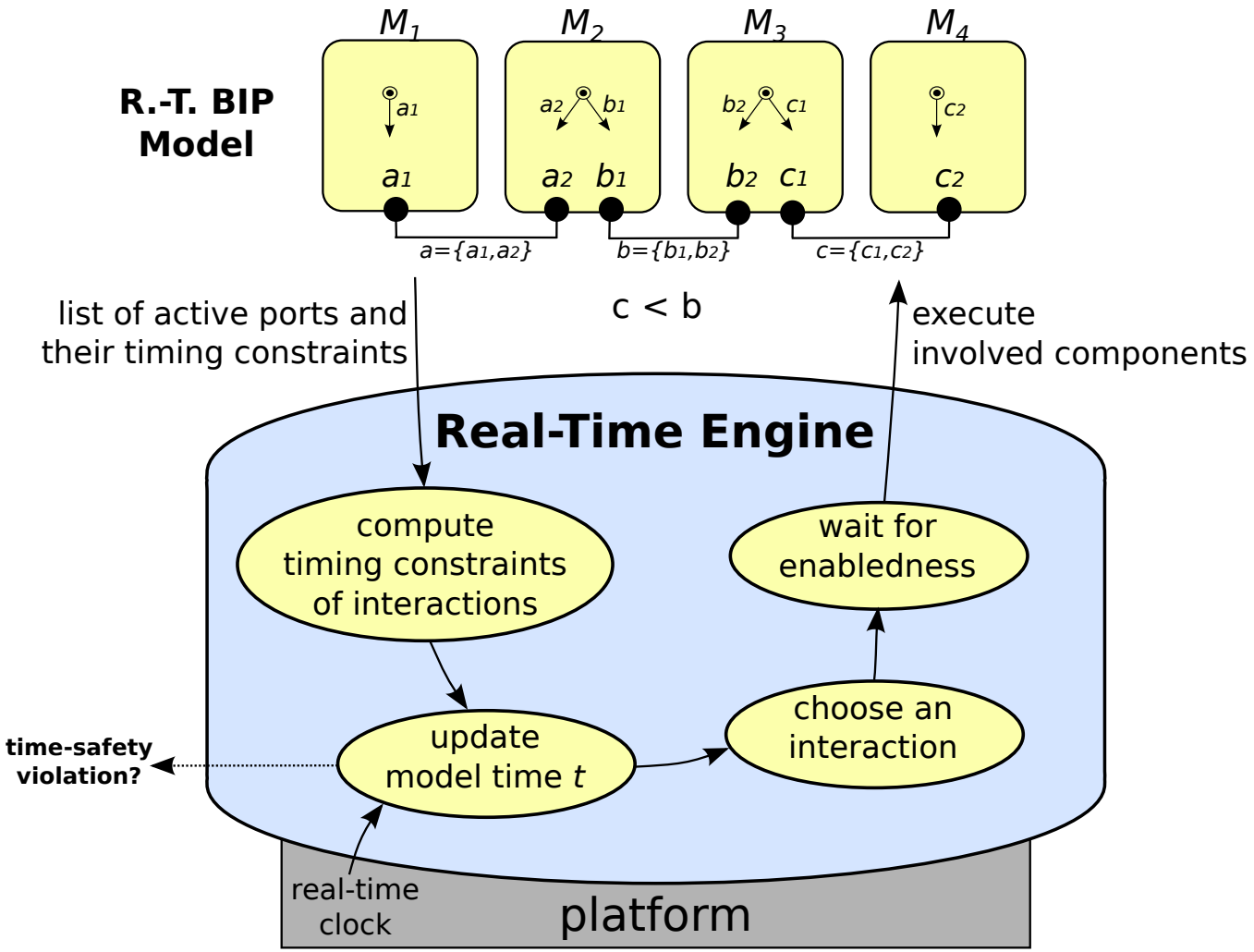
Real-Time Engine

Implements the semantics of real-time BIP:

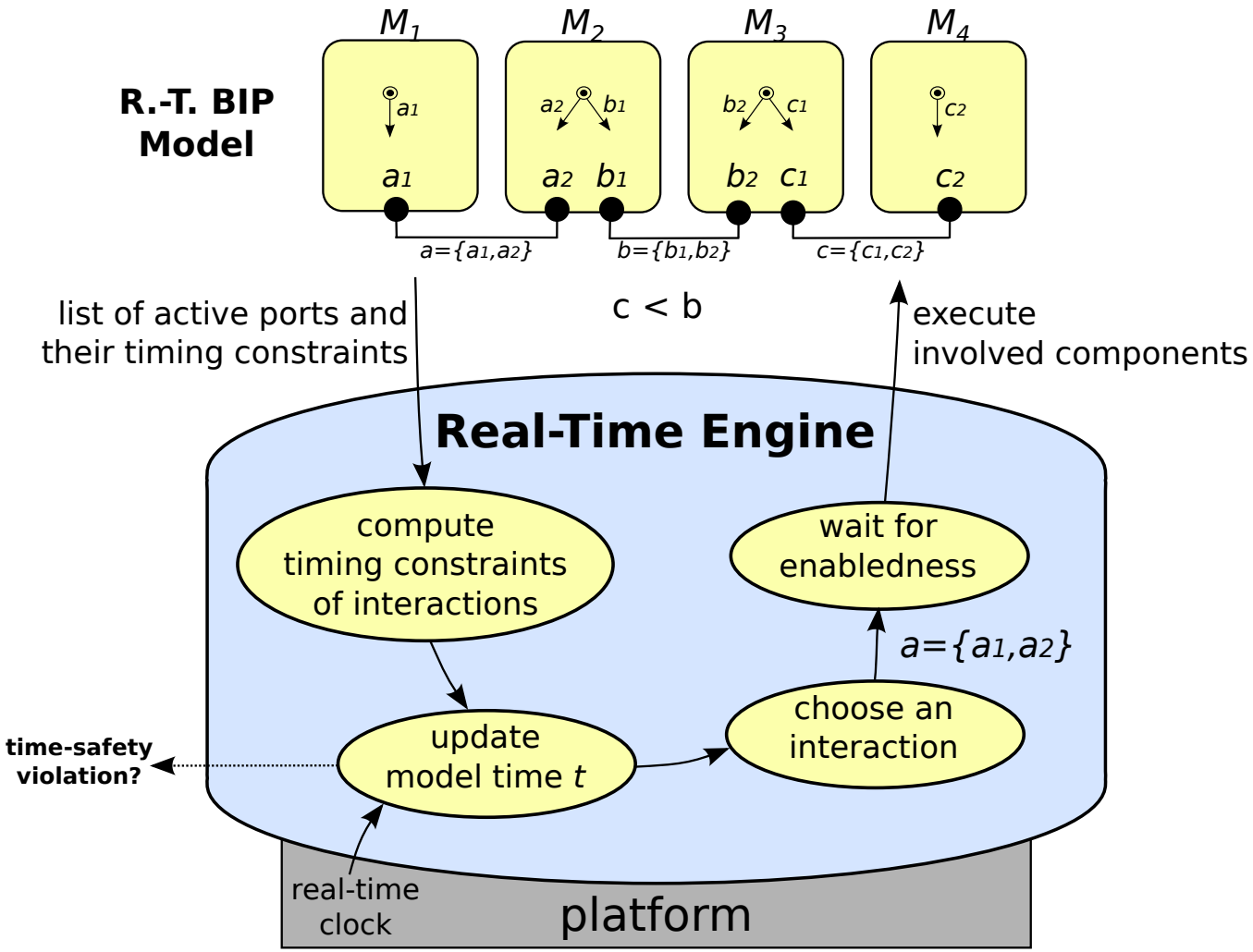
- computes on-the-fly composition of components
- use a single clock t (global time)
- t updated w.r.t. real-time after execution, and time-safety is checked
- $R[x]$: last reset of clock x, i.e. current value of x is $t - R[x]$
- $R[x] := t$ whenever clock x is reset
- timing constraints are translated:

$$L \leq x \leq U \longrightarrow L + R[x] \leq t \leq U + R[x].$$

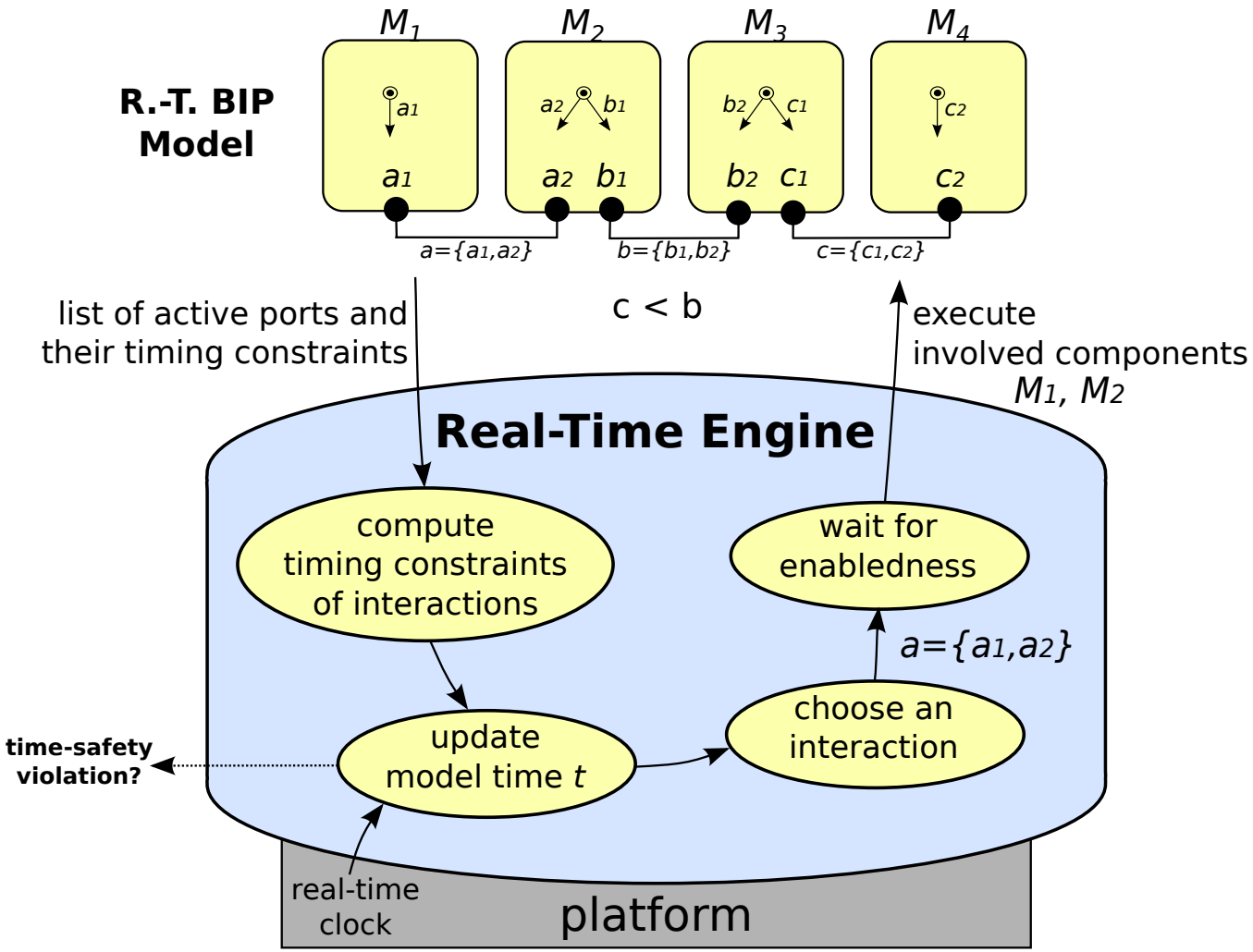
Single-Thread Engine



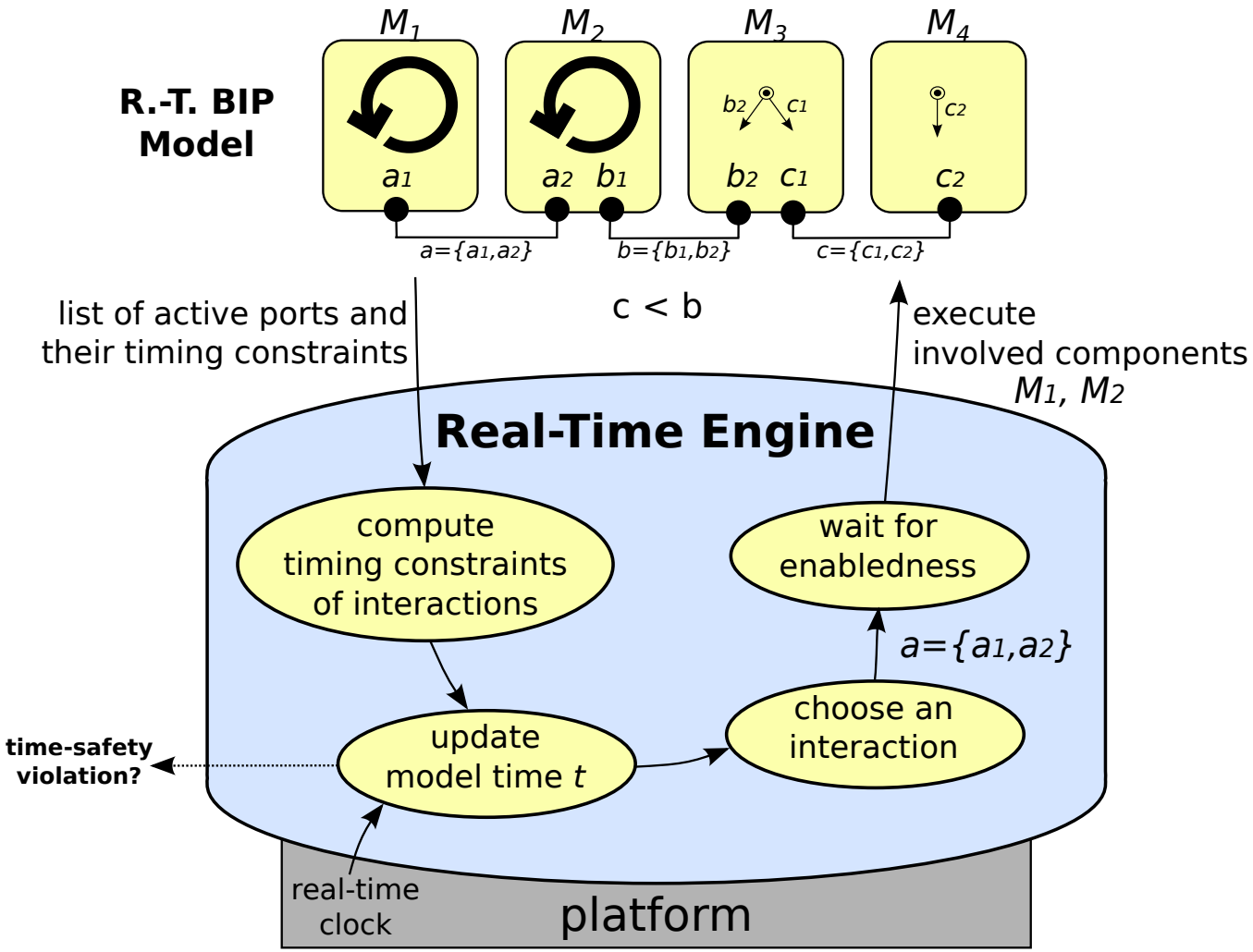
Single-Thread Engine



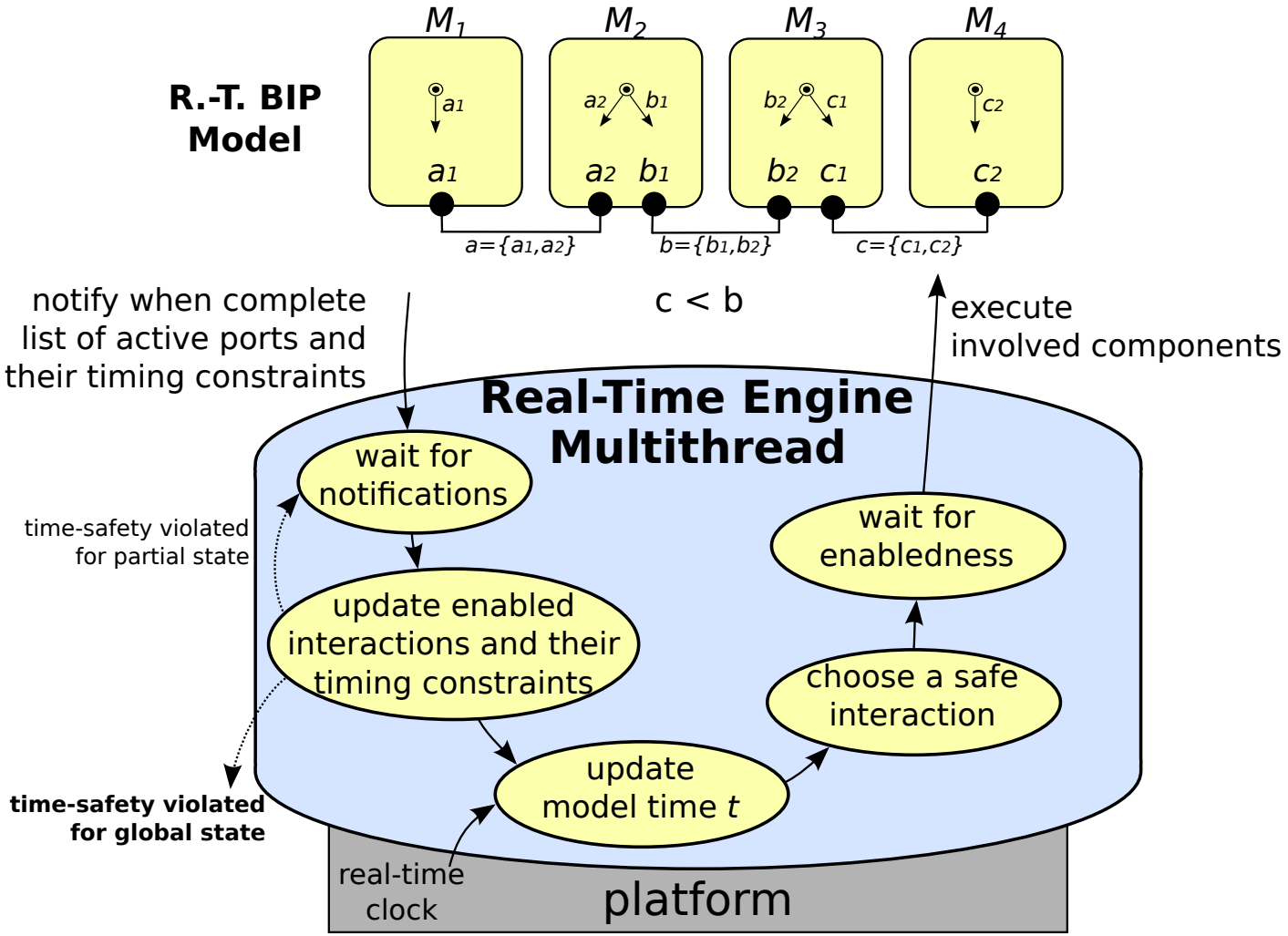
Single-Thread Engine



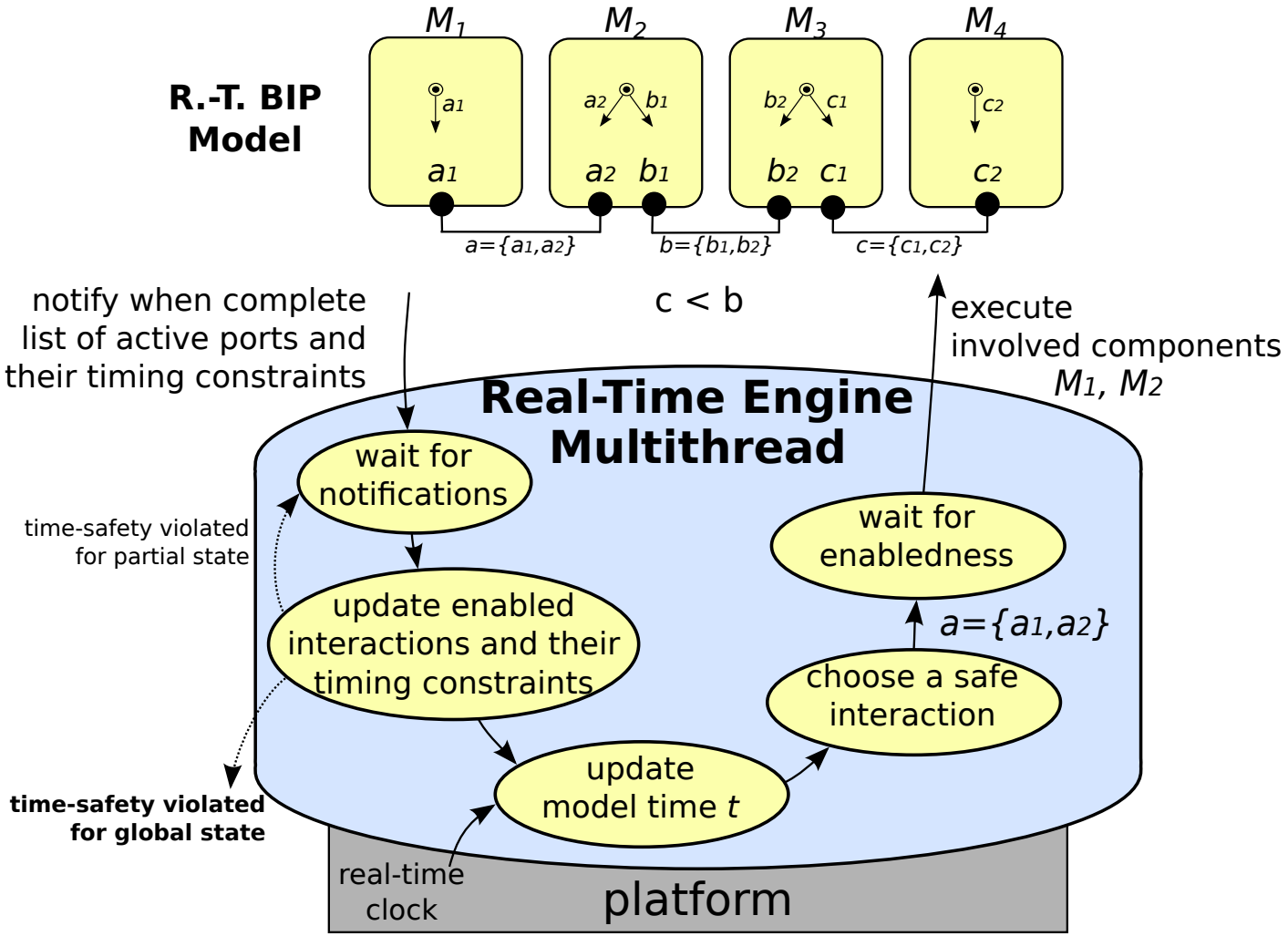
Single-Thread Engine



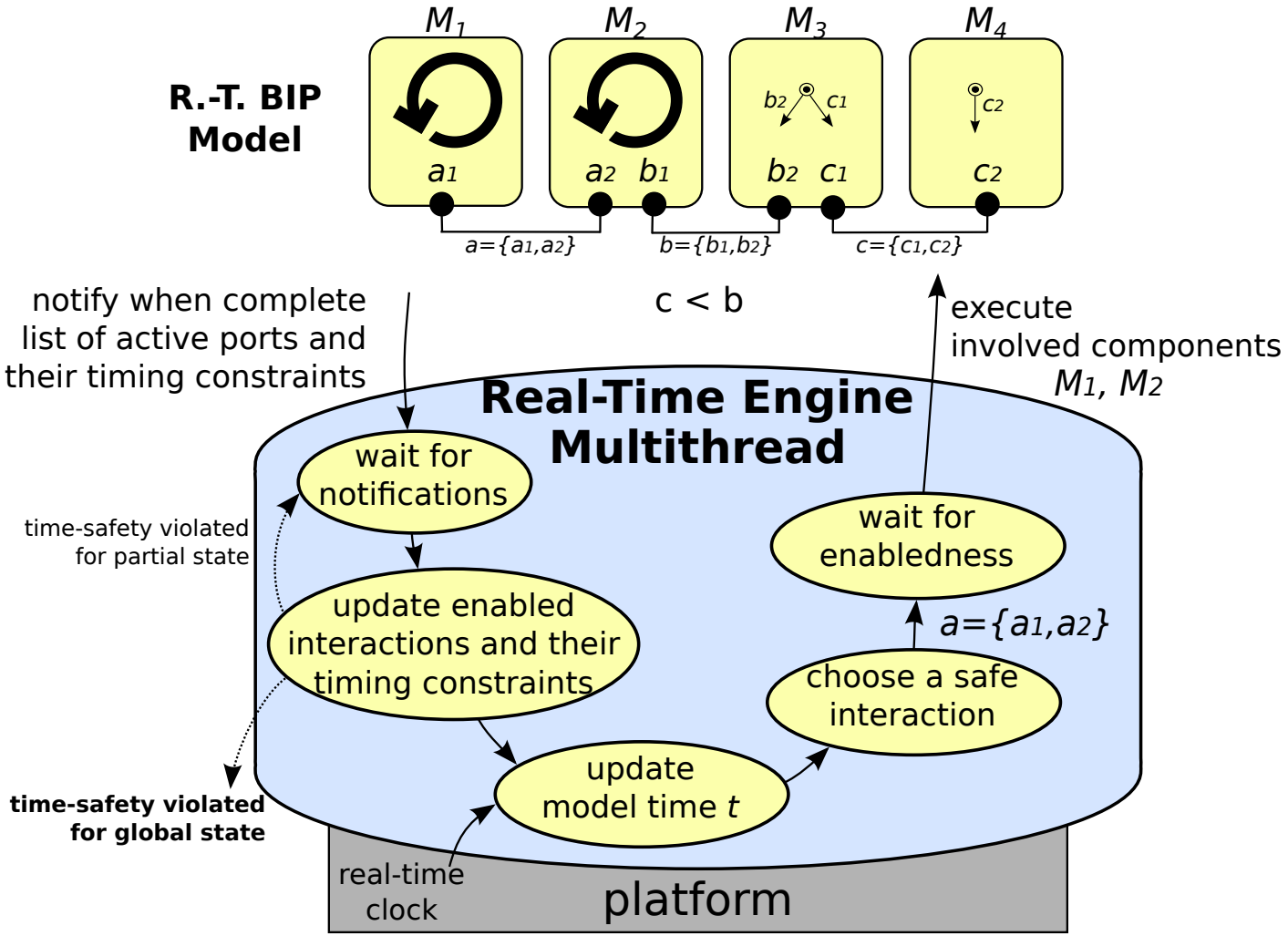
Multi-Thread Engine



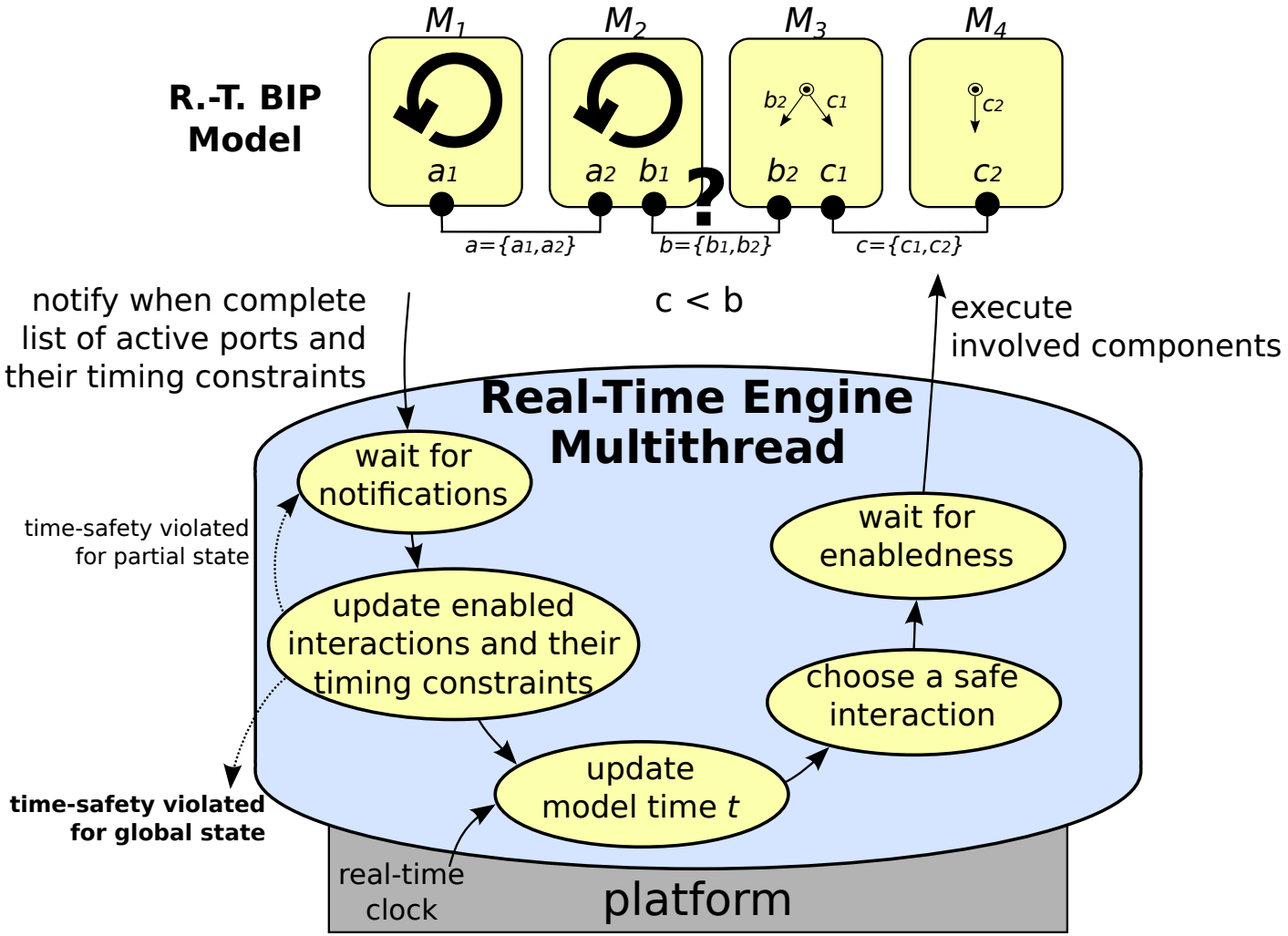
Multi-Thread Engine



Multi-Thread Engine

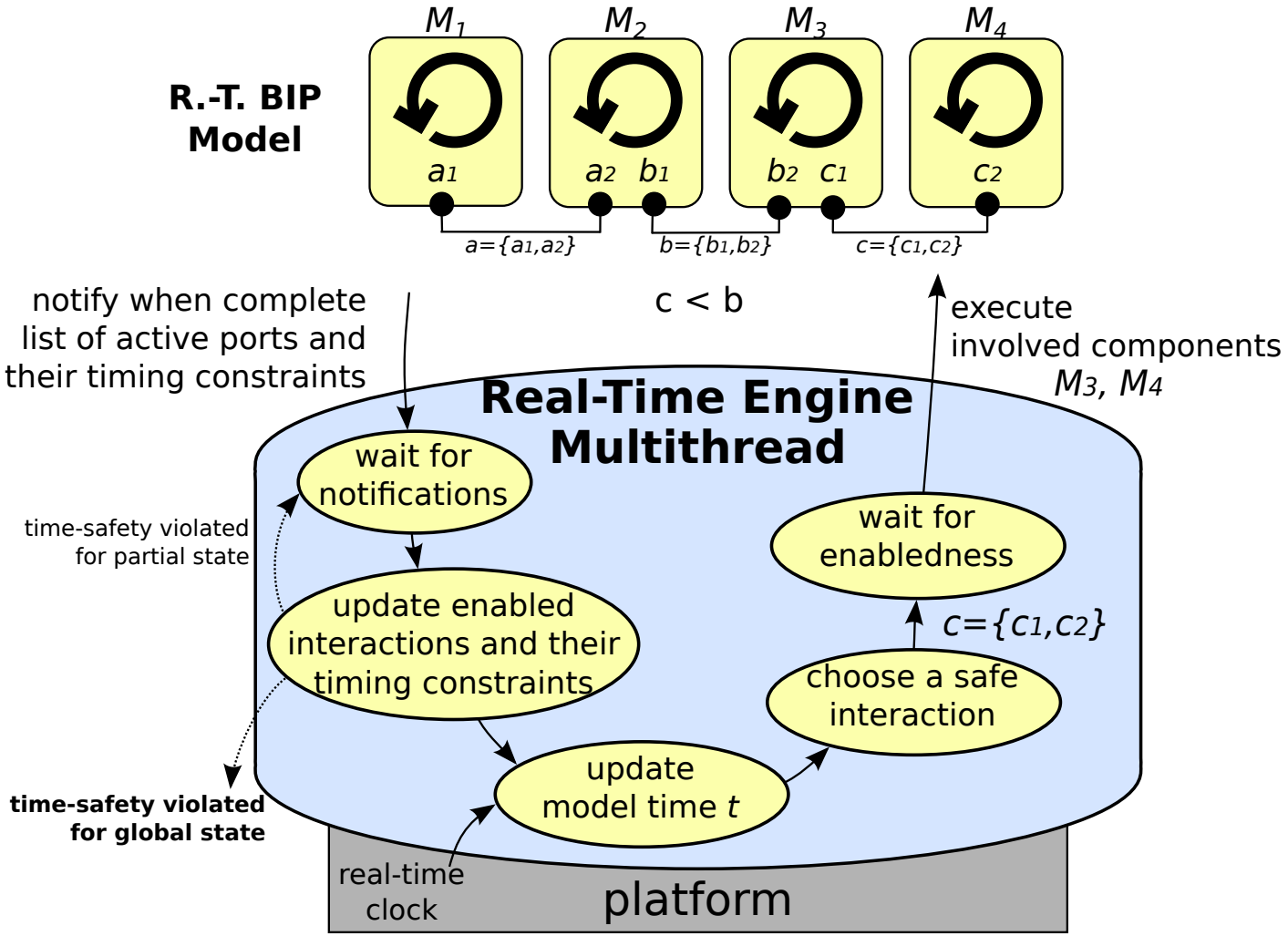


Multi-Thread Engine



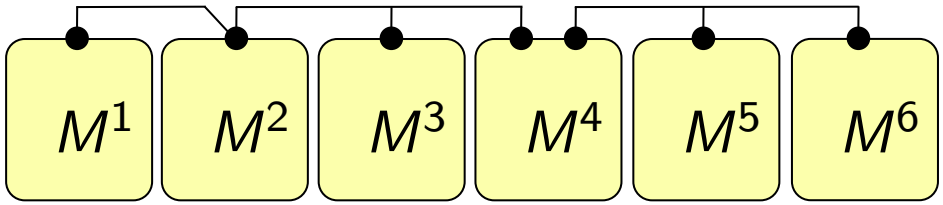
Is interaction c safe to execute, i.e. b not enabled after completion of a?
We use static code analysis to check this.

Multi-Thread Engine

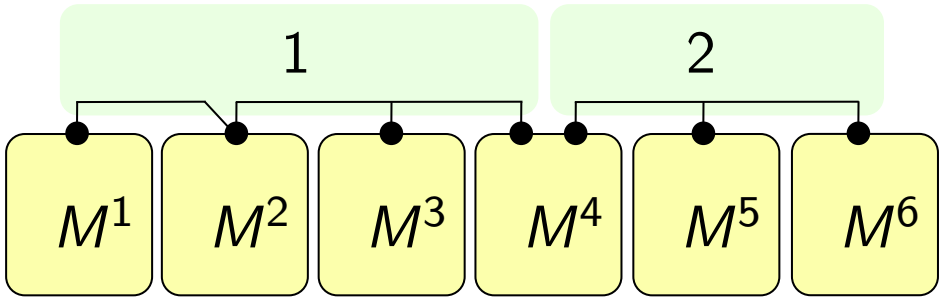


If c is safe, then execute it in parallel with a.

Distributed Real-Time BIP

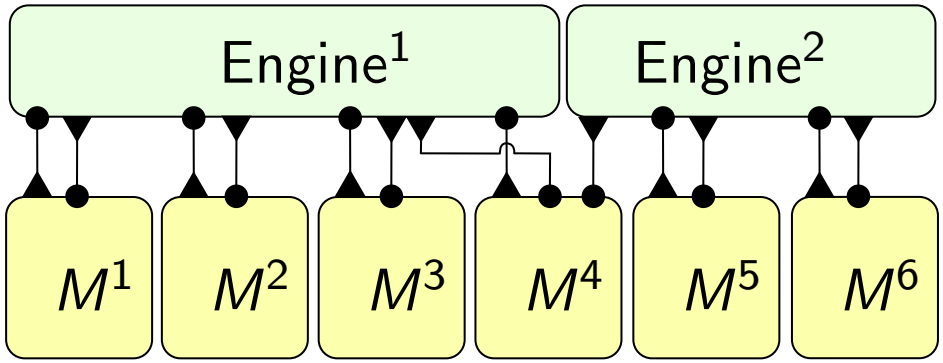


Distributed Real-Time BIP



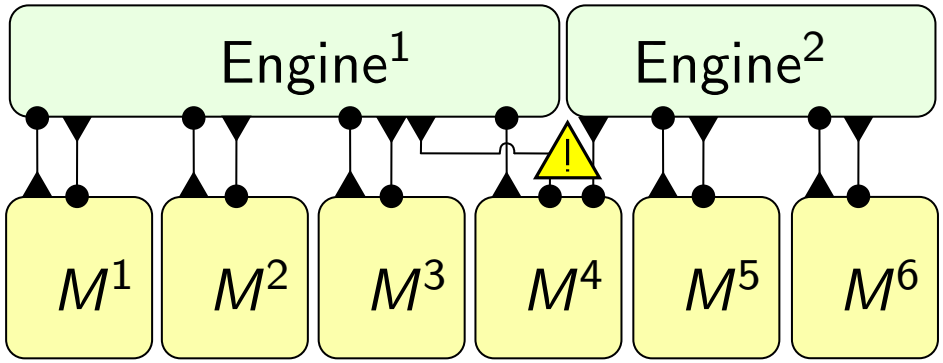
- Choose a partition of the interactions.

Distributed Real-Time BIP



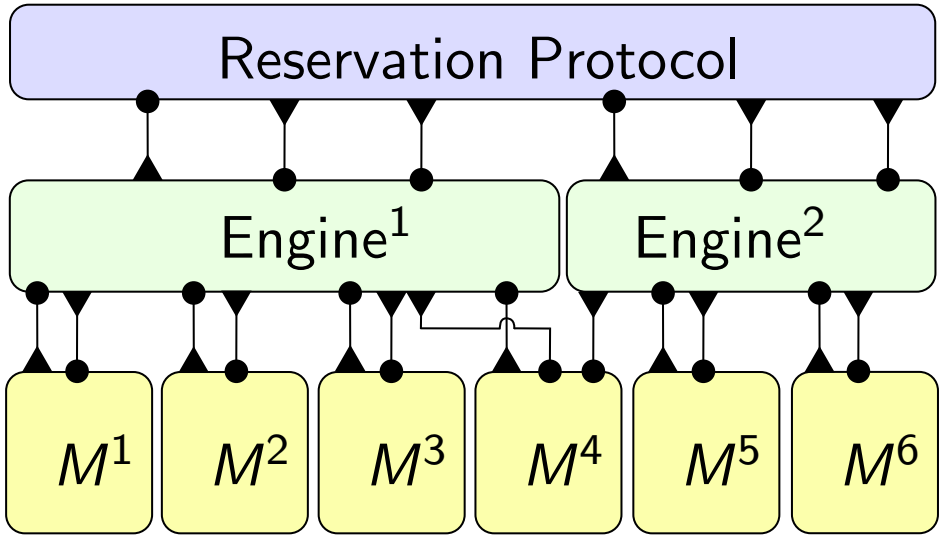
- Choose a partition of the interactions.
- Generate engines and send/receive protocols to implement interactions execution.

Distributed Real-Time BIP



- Choose a partition of the interactions.
- Generate engines and send/receive protocols to implement interactions execution.
- Conflicting situations (shared components) between engines may require additional synchronizations.

Distributed Real-Time BIP



- Choose a partition of the interactions.
- Generate engines and send/receive protocols to implement interactions execution.
- Conflicting situations (shared components) between engines may require additional synchronizations.
- We use addtional components to handle those issues.

Distributed Real-Time BIP

The challenge is to generate correct implementations in presence of:

- non uniform measure of the physical time due to multiple real-time clocks, i.e. clock are drifting
- communication delays: plan interaction execution as early as possible execution using partial knowledge of system state.

Compositional Verification of Real-Time BIP

- Given a composed system $M = \gamma(M^1, \ldots, M^n)$, safety properties can be inferred from over-approximation of reachable states:

Verification Rule

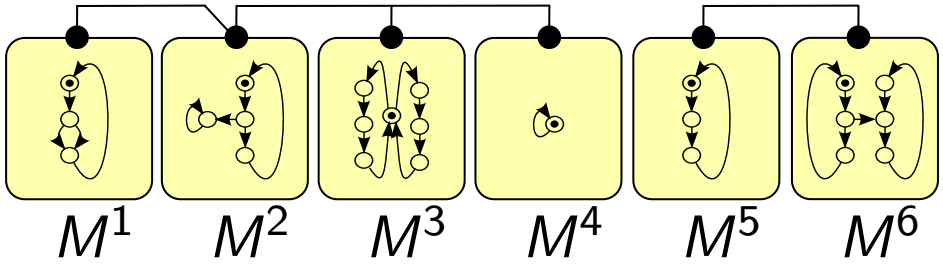
$$\frac{Reach_{app}(M) \Rightarrow P \text{ s.t. } Reach(M) \subseteq Reach_{app}(M)}{S \models \Box P}$$

- Idea: compute $Reach_{app}(M)$ based the architecture of M:

Compositional Verification Rule

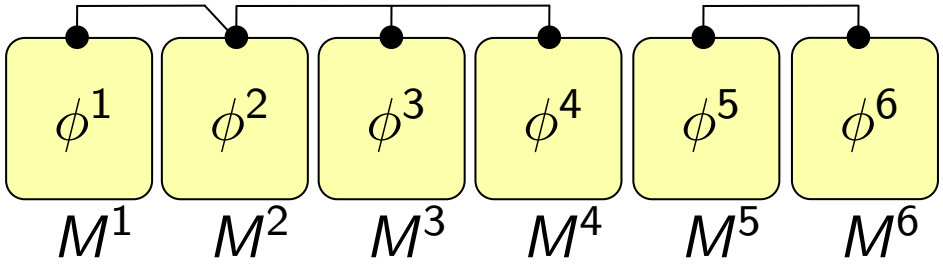
$$\frac{M^1 \models \phi^1, \ldots, M^n \models \phi^n, II(\gamma), \phi^1 \wedge \ldots \wedge \phi^n \wedge II(\gamma) \Rightarrow P}{S \models \Box P}$$

Compositional Verification of Real-Time BIP



- Add history clocks h_{a_i} for all actions a_i, reset when a_i is executed

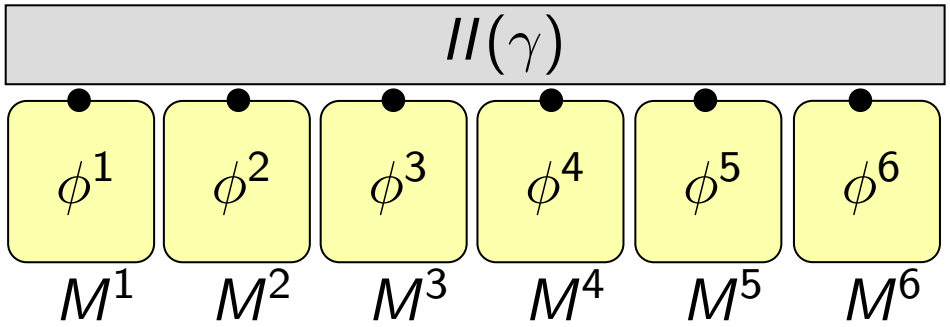
Compositional Verification of Real-Time BIP



- Add history clocks h_{a_i} for all actions a_i, reset when a_i is executed
- We propose 3 types of invariants:
  - component invariants ϕ^i: reachable states of components computed symbolically using zones:

$$Reach(q,\zeta) = \{(q,\zeta)\} \cup \bigcup_{t \in \longrightarrow} Reach(succ(t,q,\zeta))$$

Compositional Verification of Real-Time BIP

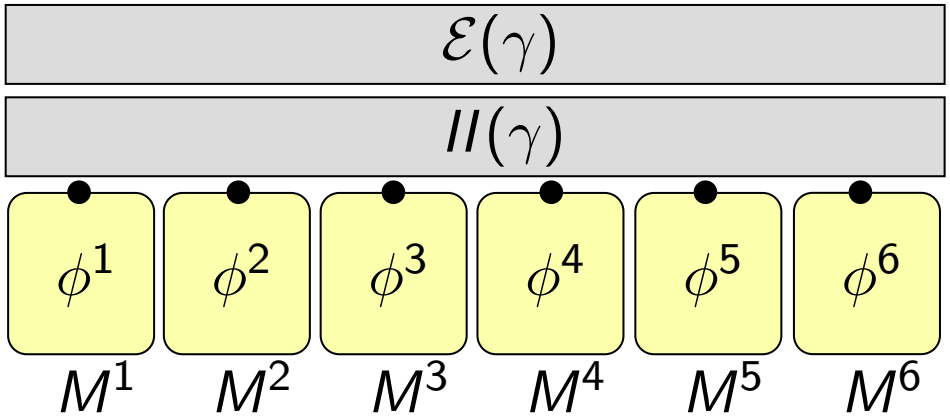


- Add history clocks h_{a_i} for all actions a_i, reset when a_i is executed
- We propose 3 types of invariants:
  - component invariants ϕ^i: reachable states of components computed symbolically using zones:

$$Reach(q,\zeta) = \{(q,\zeta)\} \cup \bigcup_{t \in \longrightarrow} Reach(succ(t,q,\zeta))$$

  - interaction invariants $II(\gamma)$: constraints on control locations induced by interactions γ

Compositional Verification of Real-Time BIP



- Add history clocks h_{a_i} for all actions a_i, reset when a_i is executed
- We propose 3 types of invariants:
  - component invariants ϕ^i: reachable states of components computed symbolically using zones:

$$Reach(q,\zeta) = \{(q,\zeta)\} \cup \bigcup_{t \in \longrightarrow} Reach(succ(t,q,\zeta))$$

  - interaction invariants $\Pi(\gamma)$: constraints on control locations induced by interactions γ
  - history clock invariants $\mathcal{E}(\gamma)$: deduced from the fact that after execution of interaction $a = \{a_i\}_{i \in I}$ all history clocks h_{a_i} are equal

Thank you.